NASA		NASA

USL / DBMS NASA / RECON

WORKING PAPER SERIES

Report Number

DBMS.NASA/RECON-9

The USL/DBMS NASA/RECON Working Paper Series contains a collection of reports representing results of activities being conducted by the Computer Science Department of the University of Southwestern Louisiana pursuant to the specifications of National Aeronautics and Space Administration Contract Number NASW-3846. The work on this contract is being performed jointly by the University of Southwestern Louisiana and Southern University.

For more information, contact:

Wayne D. Dominick

Editor
USL/DBMS NASA/RECON Working Paper Series
Computer Science Department
University of Southwestern Louisiana
P. O. Box 44330
Lafayette, Louisiana 70504
(318) 231-6308

DBMS.NASA/RECON-9		WORKING PAPER SERIES

KNOWLEDGE BASED SYSTEMS:

A CRITICAL SURVEY OF MAJOR CONCEPTS, ISSUES, AND TECHNIQUES

A Thesis

Presented to

The Graduate Faculty of

The University of Southwestern Louisiana

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

Srinu Kavi

December 1984

KNOWLEDGE BASED SYSTEMS:

A CRITICAL SURVEY OF MAJOR CONCEPTS, ISSUES, AND TECHNIQUES

Srinu Kavi

APPROVED:

----------	----------
Wayne D. Dominick, Chairman Associate Professor of Compter Science	William R. Edwards, Jr. Associate Professor of Computer Science
----------	----------
Thomas R. Cousins Assistant Professor of Computer Science	Joan T. Cain Dean Graduate School

ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. Wayne Dominick, for his invaluable time and comments.

I would like to thank Dr. William Edwards and Dr. Thomas Cousins for serving on my committee.

I would also like to express my deepest gratitude to my parents, Mr. and Mrs. Kousalya Ranga Rao, and to my brother, Dr. Krishna M. Kavi, for their constant encouragement and support.

Lastly, I would like to thank my special friend, Lin Yan, for her time and emotional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

After being in a relatively dormant state for many years, only recently is artificial intelligence (AI) - that branch of computer science that attempts to have machines emulate intelligent behavior - accomplishing practical results. Most of these results can be attributed to the design and use of Knowledge-Based Systems, KBSs (or expert systems) - problem solving computer programs that can reach a level of performance comparable to that of a human expert in some specialized problem domain [Nau, 83]. These systems can act as a consultant for various requirements like medical diagnosis, military threat analysis, project risk assessment, etc. These systems possess knowledge to enable them to make intelligent decisions. They are, however, not meant to replace the human specialists in any particular domain.

This report surveys recent work in interactive KBSs, explaining KBS concepts, issues, and KBS technology.

Basic concepts of KBSs, including the characteristics and types of KBSs, and differences between knowledge and data, knowledge and skill, and difference between an expert and a novice are presented in Chapter 2. Also in Chapter 2, a brief description of a hypothetical KBS, and various components in a KBS are presented.

In Chapter 3, various techniques used to construct KBSs are discussed in detail.

In Section 3.1, an introductory discussion is presented for origins of KBS techniques, various choices and restrictions, knowledge representation problems, knowledge representation forms, knowledge representation units, and credibility factors. Also in Section 3.1, the differences between procedural and declarative representations are discussed.

In Section 3.2, various methods for representing knowledge in KBSs are discussed. Specifically, six representation techniques - finite-state machines, programs, predicate calculus, production rules, semantic networks, and frames - are discussed in detail.

In Section 3.3, various issues and techniques related to the inference engine of a KBS are discussed. Also in Section 3.3, two performance metrics that are useful in evaluating the performance of an inference engine are described.

In Section 3.4, after providing brief introduction for workspace representation in KBSs, two techniques (HEARSAY-Blackboard and AND/OR Graph) are discussed in some detail.

In Section 3.5, various functions and types of interfaces are discussed. Also in 3.5, the knowledge acquisition process is described. Specifically the phases involved and problems associated with the knowledge acquisition process are discussed.

In Chapter 4, various tools and languages to build KBSs are

discussed.

In Section 4.1 an introduction to various tools and languages is presented. In Section 4.2, three case studies (EMYCIN, HEARSAY-III, and AGE) for KBS building tools are described.

In Chapter 5, various considerations that should be taken into account before (and during) building a KBS are presented.

Conclusions are presented in Chapter 6 and, in Chapter 7, many problems that exist in current KBSs and, hence, future areas of research are identified.

Three appendicies are provided in this report. In Appendix A, a case study of a KBS (MYCIN) is described in detail. A list of existing KBSs and brief description of those systems are provided in Appendix B. In Appendix C, a brief introduction is provided for the Japanese Fifth Generation Computer Project.

And, finally, extensive set of references are provided at the end of this report.

Chapter 2

KNOWLEDGE-BASED SYSTEMS (KBSs)

It is necessary to distinguish, at the outset, between knowledge-based systems and other computer-based systems that contain or incorporate knowledge. Almost all computer programs and systems contain knowledge of at least two kinds: knowledge about things and knowledge about what to do with things - that is, how to manipulate or transform them. A KBS can be defined in the following way: "A knowledge-based system is one in which knowledge is collected in one or more compartments (called knowledge sources) and is of the kind that facilitates problem solving (reasoning) in a single, well-defined problem domain and whose performance is comparable to that of a human expert in some specialized problem domain". (This definition is based on the definitions presented in [Barnett & Bernstein, 77] and [Nau, 83]).

From this definition, however, it is not readily apparent what distinguishes such a system from an ordinary application program. Many application programs make use of specialized problem-solving knowledge and many of them reach high levels of performance [Nau, 83]. The discussion in the next section should help make that distinction.

2.1.1 Characteristics of KBSs

Some important characteristics of KBSs (and differences with other computer-based systems) are discussed in the following sub-sections.

2.1.1.1 Organization of Knowledge

Most computer programs organize knowledge on two levels: data and program. But most knowledge-based systems organize knowledge on three levels: data, knowledge, and control.

At the data level is information about the current problem and the current state of affairs in the attempt to solve the problem.

At the knowledge base level is general knowledge about the problem domain the system is designed and built for.

At the control level are the methods (inference engine) of applying general knowledge to solve the problem.

2.1.1.2 Performance

KBSs handle real-world, complex problems which require an expert's interpretation (or expertise). The experts produce consistently high-quality results in minimal time (i.e., they show "high performance"). High performance requires that the KBSs have not only general facts and principles but the specialized ones that separate human experts from novices [Buchanan, 82]. Accurate and high quality results are shown in

many successful KBSs in restricted classes of problems.

However, currently there are no (known) formal metrics to evaluate the performance of KBSs (see Chapter 7).

2.1.1.3 Utility (or Usefulness)

Designers of KBSs are motivated to build these systems because of the demonstrated need in many application areas, in addition to constructing programs that serve as vehicles for AI research. For example, the motivation for developing the MYCIN system - a system which provides consultive advice on diagnosis of and therapy for infectious diseases, in particular, bacterial infection in the blood, bacteremia - was the need for more (or more accessible) consultants to physicians selecting antimicrobial drugs (see the case study of MYCIN in Appendix A).

On the other hand, solving the Tower of Hanoi puzzle, per se, is not a critical bottleneck in any scientific or engineering enterprise. However, in some cases, a task is chosen just because of its inherent importance. More often than not, a problem's significance for AI research is also a factor now because KBSs are still constructed by researchers for research purposes [Buchanan, 82]. Usefulness also implies competence, consistently high performance, and ease of use.

2.1.1.4 Transparency (or Understandability or Explainability)

One of the most important characteristics of a KBS is the ability to conduct an interactive dialog with the user i.e., the

user does not view KBS as a "black box". This means the system should be able to provide coherent explanations of its line of reasoning and answers to queries about its knowledge and its results, rather than simply printing a collection of orders to the user. It is not necessary that KBSs are psychological models of the reasoning of the experts. However, they must be understandable to persons familiar with the problem [Buchanan, 82].

2.1.1.5 Heuristics

Heuristics (or hunches or rules of thumb) are an essential key to intelligent problem solving because computationally feasible, mathematically precise methods are known for only a relatively few classes of problems. A large part of what a KBS needs to know is the body of heuristics that specialists use in solving hard problems, i.e., the need to reason with judgemental knowledge as well as with formal knowledge of established (or textbook) theories [Buchanan, 82]. With the above heuristic knowledge, the system provides expert-level analyses of difficult situations.

2.1.1.6 Flexibility

Another characteristic of a KBS is that it integrates new knowledge incrementally into its existing store of knowledge, i.e., a KBS provides incremental development of knowledge over an extended time by letting the developers refine old rules and add new ones.

2.1.1.7 Modularity (or Changeability)

In KBSs, there is a clear separation of the general knowledge of a problem domain and the reasoning mechanism which uses this knowledge (as was mentioned in Section 2.1.1.1, "Organization of Knowledge"). With this separation, the program can be changed by simple modification of the knowledge base, i.e., the same general system can be used for a variety of applications, essentially by "unplugging" one set of rules and "plugging" in another.

2.1.1.8 Uncertainty

Another very important and distinguishing characteristic of a KBS is its ability to reason under uncertain or incomplete information. Let us take the example of MYCIN. It takes from 12 to 24 hours to determine whether there is an organism and make a preliminary identification of its general characteristics. Another 24 to 48 hours are required to obtain specific identification and possibly even more time to determine which specific antimicrobial drug is most effective in either counteracting the organism or arresting its growth. In many cases, the infection is serious enough that treatment must be begun before all of the analyses can be completed. Therefore, any recommended therapy must be based on incomplete information.

In building KBSs with the above characteristics, researchers have found that amassing a large amount of data rather than

sophisticated reasoning techniques is responsible for most of the power of the system. Such KBSs, previously limited to academic research projects, are beginning to enter the software market place [Gevarter, 83]. Some of the application areas where KBSs are used are:

(1) Medical diagnosis.
(2) Mineral exploration.
(3) Oil-well log interpretation
(4) Chemical and biological synthesis.
(5) Military threat assessment.
(6) Planning and scheduling.
(7) Signal interpretation.
(8) Air-traffic control.
(9) VLSI design.
(10) Equipment fault diagnosis.
(11) Speech understanding.
(12) Space defense.
(13) KB access and management.

Table 2-1 lists a few of the existing systems developed for selected problem areas. A more extensive list is provided in Appendix B.

2.1.2 Types of KBSs

Most of the KBS applications fall into a few distinct types and are summarized in Table 2-2.

Table 2-1 SOME EXISTING EXPERT SYSTEMS [Nau, 83]

SYSTEM	AREA OF EXPERTISE
AQ1	Diagnosis of Plant Diseases
CASNET	Medical Consulting
DENDRAL	Hypothesizing Molecular Structure from Mass Spectrograms
DIPMETER ADVISOR	Oil Exploration
EL	Analyzing Electrical Circuits
INTERNIST	Medical Consulting
KMS	Medical Consulting
MACSYMA	Mathematical Formula Manipulation
MDX	Medical Consulting
MOLGEN	Planning DNA Experiments
MYCIN	Medical Consulting
PROSPECTOR	Mineral Exploration
PUFF	Medical Consulting
R1	Computer Configuration

Table 2-2 GENERIC CATEGORIES OF KNOWLEDGE ENGINEERING APPLICATIONS

[Hayes-Roth, et al, 83]

CATEGORY	PROBLEM ADDRESSED
INTERPRETATION	Inferring Situation Descriptions from Sensor Data
PREDICTION	Inferring Likely Consequences of Given Situations
DIAGNOSIS	Inferring System Malfunctions from Observables
DESIGN	Configuring Objects Under Constraints
PLANNING	Designing Actions
MONITORING	Comparing Observations to Plan Vulnerabilities
DEBUGGING	Prescribing Remedies for Malfunctions
REPAIR	Executing a Plan to Administer a Prescribed Remedy
INSTRUCTION	Diagnosing, Debugging, and Repairing Student Behavior
CONTROL	Interpreting, Predicting, Repairing and Monitoring System Behaviors

2.1.2.1 Interpretation Systems

Interpretation systems analyze the data or observables and infer their meaning. This category can be further divided into two: data analysis systems and situation analysis systems.

(a) Data Analysis Systems

This category includes surveillance, speech understanding, image analysis, chemical structure elucidation, signal interpretation, and oil-well log interpretation. A key requirement for these systems is to find consistent and correct interpretations of the data. It is often important that analysis systems be rigorously complete, i.e., they consider the possible interpretations systematically and discard candidates only when there is enough evidence to rule them out.

An example of this type is DENDRAL which interprets mass spectrometer data [Feigenbaum, et al, 71]. The data are measurements of the mass of molecular fragments and interpretation is a determination of one or more chemical structures.

(b) Situation Analysis Systems

This category includes analysis of electrical circuits, digital circuits, mechanics problems, earthquake damage assessment for structures, and military threat analysis. A key requirement of these systems, in addition to the requirements of

the data analysis systems, is plausible reasoning and its ability to recover from tentative assumptions.

An example of this type is system EL [Sussman, 77], which uses forward reasoning with electrical laws to compute electrical parameters (voltage and current) at one node of a circuit from parameters at other nodes.

2.1.2.2 Prediction Systems

Prediction systems infer likely consequences (i.e., forecast the course of the future) from given situations (past and present). This category includes weather forecasting, demographic predictions. traffic predictions, crop estimates, and military forecasting. A key requirement for these systems is the ability to refer to things that change over time and to events that are ordered in time. They must have adequate models of the ways that various actions change the state of the modeled environment over time.

Currently there is no known KBS which falls into this category.

2.1.2.3 Diagnosis Systems

Diagnosis systems infer system malfunctions (or disease state in a living system) from observables. This category includes medical, electronic, mechanical and software diagnosis, and diagnosis of nuclear reactor accidents. Key requirements include those of interpretation. A diagnostician must understand

the system organization (i.e., its anatomy) and the relationships and interactions between subsystems.

An example of this category is INTERNIST-1, an experimental computer based diagnostic consultant for general internal medicine. The system can deal with five hundred diseases and it is able to diagnose multiple and simultaneous diseases [Pople, 77].

2.1.2.4 Design Systems

Design systems develop specifications (or configurations of objects) that satisfy particular requirements of the design problem. They include circuit layout, building design, and chemical synthesis. Requirements for these systems include minimization of an objective function that measures costs and other undesirable properties of potential design, and the ability to explain and justify the design decisions.

An example of this type is R1, a system for configuring Digital Equipment Corporation VAX computer systems [McDermott, 80].

2.1.2.5 Planning Systems

Planning systems design actions that can be carried out to achieve goals. They include automatic programming, robotics, planetary flybys, mission planning, design of molecular genetics experiments, and military planning problems. A key requirement for these systems is that they construct a plan that achieves

goals without consuming excessive resources or violating constraints. If goals conflict, they establish priorities. Since planning always involves a certain amount of prediction, these planning systems also have certain requirements of prediction systems.

An example of this type is MOLGEN, a genetic engineering system to assist geneticists in planning laboratory experiments concerned with manipulation of DNA with restriction enzymes [Martin, et al, 77].

2.1.2.6 Monitoring Systems

Monitoring systems continuously observe system behavior, interpret the signals and set off alarms when intervention is required. The key requirements for monitoring systems are similar to those of diagnostic systems with the additional requirement that the recognition of alarm conditions be carried out in real time. For credibility, these system should avoid false alarms. Many computer-aided monitoring systems exist in nuclear power plants, air traffic control, disease, regulatory, and fiscal management tasks.

An example of this type of system is VM (Ventilator Monitor), which monitors a patient using a mechanical breathing device after surgery [Fagan, 80].

2.1.2.7 Debugging Systems

Debugging systems prescribe remedies for malfunctions, i.e., they create specifications or recommendations for correcting a diagnosed problem. The key requirements are similar to that of planning, design, and prediction systems.

Computer aided debugging systems exist for computer programming in the form of an intelligent knowledge base and text editors, but none qualify as an knowledge-based system.

2.1.2.8 Repair Systems

Repair systems create plans (or recommendations) and execute those plans to correct some diagnosed problem. The requirements for these systems are similar to those of debugging and planning systems.

Computer-based repair systems exist in automotive, network, avionic, and computer maintenance. Construction of KBSs of this type has just begun.

2.1.2.9 Instructional Systems

The computer-aided instruction systems (or, simply, instruction systems) diagnose and debug student behaviors and plan a tutorial interaction intended to convey the remedial knowledge to the student. Because these systems incorporate diagnosis and debugging subsystems, the requirements for instructional systems are similar to those of diagnosis and

debugging systems. They include electronic trouble shooting, medical diagnosis, teaching, mathematics, and coaching a game.

An example of this system is SOPHIE, which teaches problem-solving skills in the context of a simulated electronic laboratory. SOPHIE allows the student to have a one-to-one relationship with a computer-based "expert" who helps him come up with his own ideas, experiment with those ideas, and when necessary, debug them.

2.1.2.10 Control Systems

An expert control system adaptively governs the overall behavior of a system which include interpreting, predicting, repairing, and monitoring system behaviors. The requirements of these systems include those of interpretation, prediction, repairing, and monitoring systems. This category includes air traffic control, business management, battle management, and mission control.

KBSs are just entering this field.

2.1.2.11 Knowledge Acquisition Systems

These systems assist in the construction of large knowledge bases and refinement of existing knowledge by helping transfer expertise from the human expert to the knowledge base. The key requirements of these systems include organization of knowledge into meta-level knowledge which helps in the task of assembling and maintaining large amounts of knowledge and in providing a

natural language interface. This category includes maintaining large medical knowledge bases and geological knowledge bases.

An example of this type is TEIRESIAS [Davis & Lenat, 82], a system which makes possible the interactive transfer of expertise from a human expert to the knowledge base of a high performance program, in a dialog conducted in a restricted subset of natural language.

2.1.2.12 Expert System Construction Systems

This type of system provides general-purpose programming systems to build expert systems. The key requirements include provision for knowledge representation techniques and intelligent editing facilities. This category includes medical consultation systems and electronic system diagnosis systems.

An example of this type is ROSIE [Fain, et al, 81], which provides a general-purpose programming system for building expert system. This system also has very sophisticated editing facilities which check syntax and semantics of the input.

2.1.2.13 Image Understanding Systems

These systems attempt to identify and classify instances of modeled objects and, at the same time, extract three-dimensional information from a monocular image concerning the shape, structure, and three-dimensional location and orientation of the objects. The key requirements for this type of system are similar to interpretation, prediction, modeling, and description

systems. This category includes aerial photography interpretation and views of automated assembly work-stations.

An example of this type is the VISIONS system, which has been tested with outdoor scenes [Cohen & Feigenbaum, 82].

2.1.2.14 Concept-Formation Systems

Currently only one system of this type exists: AM. AM models one aspect of elementary mathematics research: developing new concepts under the guidance of a large body of heuristic rules [DaviS & Lenat, 82].

2.1.3 Knowledge and Data

The concept of knowledge itself is not simple, in the sense that it can be rigorously defined or bounded, nor it can be divorced from the means of acquiring or using it. The latter is equally true whether we are speaking of human or computer based knowledge-based systems. However, some simple observations can be made about knowledge and data.

Widerhold [Widerhold, 84] observes that:

(1) Knowledge considers general aspects of data.

(2) Knowledge is significantly smaller than data.

(3) Knowledge does not vary rapidly (compared to data)

The following simple examples illustrate the difference between knowledge and data [Widerhold, 84]:

Mr. Lee's age is 43 years	- Data
Middle-age is 35-50	- Knowledge
People of middle-age are careful	- Knowledge
Mr. Lee has never had a traffic accident	- Data

2.1.4 Knowledge and Skill

Webster's dictionary defines skill as "the ability to use one's knowledge effectively and readily in execution or performance". Skills refer to organized modes of operation and generalized techniques for dealing with problems. The problems may be of such nature that little or no specialized and technical information, thus no special knowledge, is required. Other problems may require specialized and technical information at a rather high level such that specific knowledge is required in dealing with the problem [Barnett & Bernstein, 77].

The main characteristic of a skilled performance include great speed, or other efficiencies, reduced error, reduced cognitive load (attentional requirements) and increased adaptability and robustness [Hayes-Roth, et al, 83].

2.1.5 Expert and Novice

The difference between expert and novice - experts solve complex problems considerably faster and with less errors than novices - are commonplace within everyday experience. During the past decade, substantial progress has been made in exploring and

explaining the human information processes that underlie expert performance.

The major components of an expert's skill (expertise) which separates the expert from the novice are: perceptual knowledge, recognition capabilities, and the way in which information is represented in long-term memory.

An expert knows a great many things and can rapidly evoke particular items relevant to the problem at hand. Although a sizable body of knowledge is prerequisite to expert skill, that knowledge must be indexed by a large numbers of patterns that, on recognition, guide the expert in a fraction of a second to relevant parts of the knowledge store.

Human memory consists of a complex organization of nodes connected by links called "list structures". Human long-term memory can be represented formally by such node-link structures and almost all computer simulations of cognition use list structures together with productions that act on these list structures as their fundamental means for representing memory. These formalisms capture the associative properties of long-term memory. An excellent discussion on expert and novice (on which the discussion above was based) can be found in [Larkin, et al, 80].

2.1.6 KBSs and Expert Systems

KBSs contain large amounts of varied knowledge, which they

use during a problem solving activity. Expert systems (ESs) are a species of KBSs, which use large amounts of knowledge and whose performance is equivalent to that of an expert in a given domain.

Expert performance means, for example, the level of very experienced engineering or scientific tasks, or very experienced MD diagnosing and recommending therapy. The ES acts as an intelligent assistant to a human expert.

A system that can understand images, or understand speech, may rely on a large knowledge base to achieve its perceptions, but it does not require any human expertise. For instance, normal human beings are born with eyes, ears, and the equipment behind them to process the signals those organs receive, and they quickly acquire the knowledge needed to understand the signals. But normal humans are not born to knowing, for example, how to diagnose a disease; that takes expertise, learned over a long period.

This report will not rely upon the formal distinction between KBSs and ESs, but will rather use the terms as though they were synonymous, and, in fact, in most of the literature these terms are used interchangeably.

2.2 A Hypothetical KBS

The following is a brief description and example of a simple, hypothetical KBS application that illustrates most of the capabilities of a KBS. This example and description is based on [Barnett & Bernstein, 77].

The hypothetical system is an automotive service consultant whose primary purpose is to help ensure the best service at the least cost for automobiles brought to a service agency. Based on the discussion in Section 2.1.2, this KBS falls into the category of repair systems and all repair systems will also have diagnosis subsystems.

The problem domain for this KBS is well bounded and well defined. Hence reasoning skills required by such a KBS are relatively simple. The knowledge required, however, is large because of the number and variety of automotive subsystems involved and the high degree of their interdependence. In addition, the system also contains a data base. The data base of the system would contain a description of each car serviced at the agency. The descrption would contain the configuration of the vehicle's engine, running gear, and optional and special equipment as well as the year of manufacture, model, color, service history, etc.

Like all KBSs, this hypothetical automotive consultant system contains two kinds of knowledge:

(1) Generic knowledge about the major elements of the vehicle (such as its cooling, ignition, and fuel systems).

(2) Knowledge about the model and its specific options.

The intent of the system is to recommend:

(1) The least expensive tests that will produce the greatest amount of specific information about the cause of the problem.

(2) The least expensive repairs for the problem.

(3) The best post-repair testing to verify that the problem is solved.

The following is a hypothetical scenario of such a system in operation. The users are the service representative and the mechanic(s) performing the test and repair work. The user language is a stylized English designed to convey a maximum amount of information between the user and the system with a minimum amount of effort.

System: Tuesday, May 3, 1977, 08:32:44

Service rep: Next customer

System: Enter owner's name or vehicle ID number

Service rep: 1743 HBAQ 8197

System: Owner is S.B. Franklin, car is 1974 Ford Gran Torino 4-dr sedan with 351-2V engine, A/C, P/S, P/B and CA smog controls. Last service - April 4, 1977 at 28,412 mi R&R water pump and lower radiator hose, what is present mileage?

Service rep: 29285

System: Please inform owner that the following regular maintenance services are due:

Change oil and filter - last changed on Jan 13, 1977 at 24,842 mi.

May we be of other service?

Service rep: Battery not keeping charge, barely able to start car this a.m.

System: Any other symptoms or problems?

Service rep: No

System: Check belt tension on alternator - most likely cause.

Service rep: Why?

System: Last service required removal of fan and belts, very likely that tension was not properly set on reinstallation.

Service rep: More (meaning more things to be checked or tested)

System: Check fluid level in battery - next most likely cause - overcharged battery. If level is low, check alternator output and voltage regulator.

Service rep: Replace battery?

System: No, battery is 6 months old and should take recharge.

Service rep: Thank you. (Signal to system that session is ended for now)

System: Car assigned number 287 in control system. First test to be performed by J.K. Uzzi in Bay 17.

After the car is moved to the mechanic's work area and the prescribed tests are performed, the following interchange takes place between the mechanic and the system:

Mechanic: JKU 17, 287 (indicating mechanic, bay and car being serviced)

System: 1974 Gran Torino, test results please.

Mechanic: Belt tension OK, battery level OK, charge low, alternator, VR output OK. Problem not found.

System: Test for above-normal current flow with ignition off. (Mechanic attaches ammeter and takes reading).

Mechanic: Reading above normal.

System: Check for faulty switch in hood light, trunk light, and courtesy lights.

Mechanic finds faulty switch in trunk light such that light never turns off.

Mechanic: Fault trunk light switch R&R. Current reading normal. Thank You.

System: You are welcome. Bye.

Some of the salient features of this hypothetical KBS are:

(1) Separation of generic knowledge from specific knowledge.

(2) The dialog between the system and the user (mechanic or service representative) has the flavor of naturalness.

(3) Expertise.

For a system to have suggested checking the belt tension of the alternator, it would have to know that the earlier removal of the belt could be related to the present problem, that the severity of the problem would depend on how poorly the tension was adjusted, and that the one month and about 900 miles before appearance of symptoms (battery failure) is not unreasonable. Since it is a highly probable cause and the easiest to test, it ranks as the first suggestion. By requesting more information, the service representative can tell the owner what else may be required and what will not likely be required such as a new battery.

2.3 KBS Components

Figure 2-1, based on [Barnett & Bernstein, 77] and [Hayes-Roth, et al, 83], shows an idealized representation of an KBS. A KBS is composed of four components (or modules):

(1) A Knowledge Base
(2) An Inference Engine
(3) An Interface
(4) A Workspace

The knowledge base contains the knowledge sources (rules

and information about the current problem, etc.) and fact files.

The inference engine (also called cognitive engine) performs the system's problem solving (inference-making or reasoning) operations. It contains procedures that manipulate knowledge contained in the knowledge base.

The interface provides problem-oriented, interactive communications between the user and the KBS. This interaction is usually in some restricted variant of English and in some cases via means of a graphics or intelligent editor.

A workspace (also called blackboard) records intermediate hypotheses, decisions, and results that a KBS manipulates during a problem-solving activity.

2.4 Knowledge Base

The knowledge base (KB) of a KBS contains knowledge sources (KSs) and fact files.

2.4.1 Knowledge Sources

A knowledge source contains rules, stipulations of the existence or non-existence of certain things, simple equivalence relationships, relationships between the concrete and abstract, knowledge of conventions about the domain, methods of the domain, etc. In other words, the breadth of knowledge acquired by one who has become expert in solving problems in the domain for which the KBS is designed [Barnett & Bernstein, 77].

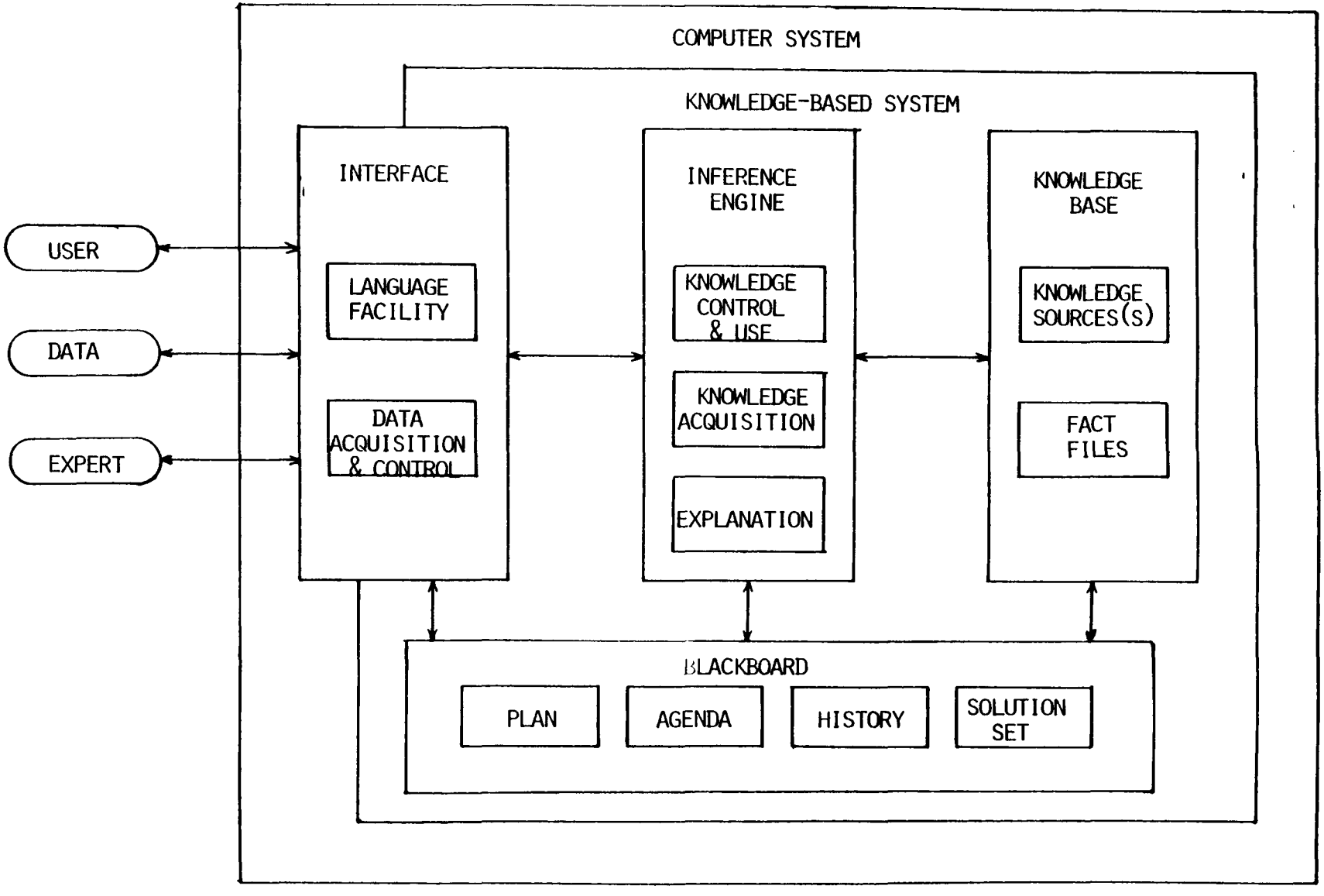


FIGURE 2-1. KBS ELEMENTS AND THEIR RELATIONSHIP
BASED ON [HAYES-ROTH, ET AL, '83] AND [BARNETT & BERSTEIN, '77]

In a KBS, it is logical to separate knowledge into different knowledge sources because :

(1) In any problem domain, each expert acquires different problem-solving knowledge and there is no efficient single method for representing all of the knowledge.

(2) In any problem-solving activity, two types (or levels) of knowledge is involved: problem-specific knowledge and how to use this knowledge. The latter is usually called "meta knowledge".

2.4.2 Fact Files

Fact files contain "hard" data such as values, attributes, etc. (for example, the contents of an engineering handbook) and, in this sense, it is equivalent to a data base. Fact files are required for the complete solution of a problem. A collection of fact files without a knowledge source is not a knowledge base. A MIS constructed from a conventional data management system is not a KBS [Barnett & Bernstein, 77], because it does not have reasoning or inferencing capability.

2.4.3 Types of Knowledge

Even though KBSs were and are being developed for a variety

of applications (see Section 2.1.2, "Types of KBSs"), the knowledge in KSs in those systems generally falls into the following five types [Barnett & Bernstein, 77]:

(1) Methods specifying cause-effect relationships, implications, or inferences depending on the richness of the relationship to be represented. Production rules, predicate calculus expressions, and other logical methods were used for representation of this type. Diagnosis systems typically use this type of knowledge. For example, MYCIN uses simple IF-THEN form of rules.

(2) Plans of action for how one would achieve an end result in the world external to the model that the system represents. For instance, in a robotic system a procedure may describe how to assemble parts of an automobile engine or, simply, how to put a block on top of another.

Models of agent behavior to infer the effects of the planning agent activities are used for representation of this type. Planning systems typically use this type of knowledge.

NOAH, a robot planning system is an example of this type and is described in [Sacerdoti, 75].

(3) Declaratives that identify objects within the modeled domain and distinguish them from objects that are not within the domain. These declaratives may describe properties of objects, relationships among objects, definitions of terms or constructs, schemata that identify the legal relationships or transformations applicable to the domain.

Semantic networks are used for representation of this type of knowledge. Interpretation and diagnostic systems typically employ this type of knowledge.

For example, CADUCEUS consists of an extremely large semantic network of relationships (approximately 100,000 associations) between diseases and symptoms in internal medicine [Pople, 81].

(4) Meta properties, which are a higher level of abstraction about the domain and the solution space and methods. Meta properties (or meta knowledge) provide means for determining and assuring the consistency, coherency, and reliability of intermediate results and steps as well as the final solution and answers.

Production rules of the IF-THEN type use this type of knowledge as well as knowledge acquisition systems.

An example of latter type is TEIRESIAS. TEIRESIAS uses meta knowledge to transfer expertise from a human

expert to the knowledge base of a high-performance program (MYCIN) in a dialog conducted in a restricted subset of English [Davis & Lenat, 82].

(5) Advice (sometimes called heuristics) that is similar to meta properties in intent, but that does not carry the same strength of influence. This is the "soft" knowledge that experts acquire from experience in working in the domain and is rarely contained in textbooks and papers [Barnett & Bernstein, 77].

The techniques used to construct these types of knowledge are discussed in Chapter 3, "Techniques Used to Construct KBSs".

2.5 Inference Engine

The inference engine (IE) provides central control of the KBS and thus affects both the performance and power of the system. The functions of an IE can be broadly divided into three categories: knowledge use and control knowledge acquisition, and explanation.

2.5.1 Knowledge Use and Control

An IE performs the system's problem solving operations. This includes inference making or reasoning, and searching. An

IE contains procedures that combine and organize (i.e., manipulate) the contents of a knowledge base. Thus, an IE acts as a manager of a knowledge base.

A small portion of knowledge in a KBS usually resides in the IE (for reasons of efficiency). The knowledge contained in the IE may be general knowledge or meta knowledge (knowledge about a knowledge base).

2.5.2 Knowledge Acquisition

Another function of the IE is to provide the mechanisms that facilitate the acquisition of new knowledge, the modification or refinement of existing knowledge, and deleting erroneous or useless knowledge, and maintaining consistent representation - all of which are done in cooperation with the expert.

2.5.3 Explanation

Another important function of the IE is to provide an explanation for its actions and its reasoning process with respect to an interaction with the user or to a solution it produces. In general, it answers questions about why some conclusion was reached or why some alternative was rejected. This explanation capability of the IE depends on the contents of the KB, information about the current problem, and prior interactions with the user.

The explanation of the IE is related only to its past

activity; the system cannot explain how it might deal with a hypothetical case or how it will continue in solving a present problem [Barnett & Bernstein, 77].

A KBS's ability to solve a particular problem depends on:

(1) How many paths there are to a solution.
(2) The ability of the IE to reduce the number to a minimum.
(3) The knowledge in the KB.
(4) What information is available within the problem statement.

Therefore, although the IE is in command and acts as the driving element, the path to a solution, and the criteria for when to accept a solution or abort a particular path are highly dependent on the content of the KB and the problem data. That is why researchers have found that "amassing a large amount of data rather than sophisticated reasoning techniques is responsible for the power of the system" [Gevarter, 83].

2.6 Interface

The interface is the communication port between the system and the outside world. Based on the functions provided, the interface of a KBS can be viewed as three different interfaces:

user interface, knowledge acquisition (expert) interface, and data interface. Each one is discussed in the following sub-sections.

2.6.1 User Interface

The user interface provides the necessary facilities for the user as a poser of problems and consumer of results (answers and justifications or explanations). The user interacts with the interface in a jargon specific to the domain of the KBS and usually in some restricted variant of English (and sometimes via means of a graphics or intelligent editor). Thus, the user interface acts as a language processor. Typically, the language processor parses and interprets user questions, commands, and volunteered information. Conversely, the language processor formats information generated by the system, including answers to questions, explanations and justifications for it's behavior, and requests for data.

Existing KBSs generally employ natural language parsers written in INTERLISP to interpret user inputs, and use less sophisticated techniques exploiting canned text to generate messages to the user [Hayes-Roth, et al, 83].

2.6.2 The Knowledge Acquisition Interface

The knowledge acquisition (KA) interface (also known as

expert interface) is used by a domain expert (who has gained some feeling for the system) as the provider of knowledge for the KSs. Associated with the KA interface is some means of verifying the incoming knowledge, sometimes limited to syntax checking, but often including tests for coherence and consistency with prior knowledge both in the KSs and the IE.

The knowledge acquisition process is discussed in more detail in Section 3.5.4, "Knowledge Acquisition Process".

2.6.3 Data Interface

The data interface is similar to that of most other interactive computer systems in that it incorporates:

(1) Facilities for user input of parameters, data, and responses to the system's queries.

(2) The mechanism for locating and accessing files or data bases.

Many of the functions necessary to provide the data interface may be drawn directly from the computer system environment within which the KBS functions.

2.7 Workspace Representation

Workspace (also known as "blackboard") records intermediate hypotheses, decisions, and results that a KBS manipulates during a problem-solving activity, i.e., it is the encapsulation of the system's current state in a problem solving activity. It includes plan, agenda, history, and solution set.

2.7.1 Plan

A plan describes the overall or general attack the system will pursue against the current problem, including current plans, goals, problem states, and contexts.

2.7.2 Agenda

An agenda is a list of activities that can be done next which generally correspond to knowledge base rules that are relevant to some decision taken previously.

2.7.3 History

History records what has been done (and why) to bring the system to its current state, which is used to provide explanations.

2.7.4 Solution Set

A solution set represents the candidate hypotheses and decisions the system has generated thus far, along with the dependencies that relate decisions to one another.

2.8 Separation of KBS Components

The separation of the elements of a KBS is a necessary condition for including a system in that category, since it permits the changing of the domain of application by extending, expanding, or substituting another KB independently of the inference engine [Barnett & Bernstein, 77].

Several researchers have illustrated the generality of their systems by showing that they can be applied to another domain merely by removing the rules for a given domain (i.e., knowledge base) and substituting rules for the new one [Van Melle, 79], [Goldberg & Weiss, 80].

For example EMYCIN is the inference engine of MYCIN, to which several different knowledge bases have been experimentally attached for solving different classes of problems.

Every domain, however, has its own peculiarities. Despite the good intentions of system builders, these peculiarities inevitably influence the design of a system. As a result, a serious attempt to build a KBS almost always changes in all parts of the system [Duda & Gashing, 81]. Recognizing this, many

researchers have recently begun developing tools or languages for constructing KBSs. They are discussed in Chapter 4, "KBS Building Tools and Languages".

2.9 Summary

In summary, to qualify as a KBS, a system must [Barnett & Bernstein, 77] :

(1) Be externally invoked by an expert in the domain of applicability.

(2) Have an identifiable IE that reasons plausibly using the KB and whose solution path is controlled by the content of the KB and problem data.

(3) Have the potential for explaining its behavior.

(4) Have an identifiable KB that contains expert domain-specific knowledge (this is the most critical aspect of a KBS).

(5) Be organized and structured so that its KB can be expanded and extended and the system's performance improved.

Chapter 3

TECHNIQUES USED TO CONSTRUCT KBSs

3.1 Introduction

3.1.1 Origins of KBS Techniques

Since the mid-60's, there has been a major shift in AI research. The shift was from a search for broad, general laws of thinking toward an appreciation of specific knowledge - facts, experiential knowledge, and how to use knowledge - as the central issues in intelligent behavior [Feigenbaum & McCorduck, 83]. A direct result of this shift (called "applied AI") is construction of KBSs or expert systems. Thus, AI techniques are widely used in KBS construction. In addition to AI, several other computer science areas have developed techniques that are used in the construction of KBSs. A summary of contributors and techniques is shown in Table 3-1.

For example, language processing techniques - specifically, parsing and understanding, question and response generation, knowledge representation and acquisition - are used for the interface component of KBSs.

Table 3-1 ORIGINS OF KBS TECHNIQUES
(Based on [Barnett & Bernstein, 77])

ARTIFICIAL INTELLIGENCE (AI)

- Heuristic Search
- Inference and Deduction
- Pattern Matching
- Knowledge Representation and Acquisition
- System Organization

LANGUAGE PROCESSING

- Parsing and Understanding
- Question and Response Generation
- Knowledge Representation and Acquisition

THEORY OF PROGRAMMING LANGUAGES

- Formal Theory of Computational Power
- Control Structures
- Data Structures
- System Organization
- Parsing

MODELING AND SIMULATION

- Representation of Knowledge
- Control Structures
- Calculation of Approximations

DATA BASE MANAGEMENT

- Information Retrieval
- Updating
- File Organization

SOFTWARE ENGINEERING

- System Organization
- Documentation
- Iterative System Development

APPLICATION AREAS

- Domain-Specific Algorithms
- Human Engineering

Similarly, data base management techniques - specifically, information retrieval, updating, file organization - are used for the knowledge base component of KBSs.

3.1.2 Choices and Restrictions

Figure 3-1 (a modification of [Barnett & Bernstein, 77] p. 4.3) illustrates the relationships between choices and restrictions in building KBSs. The left hand side (lhs) of the dotted line in Figure 3-1 shows domain specific items (or choices) and the right hand side (rhs) shows available techniques (or restrictions).

For example, in any problem domain, the expert's available knowledge model necessarily limits (or restricts) the choices for representing knowledge in a KB. Similarly, the expert's reasoning principles and methods directly affect (or restrict) methods that can be used to build an IE in a KBS.

Likewise user expectations dictate (or at least influence) explanation facilities.

Figure 3-1 also illustrates another interesting point: relative importance of choices in a KBS. According to Barnett & Bernstein [Barnett & Bernstein, 77], domain considerations are most important followed by choices of KB representation. Everything else is of less importance. Whether this is a fact or a practice is not certain. However, many existing KBSs confirm this view [Hayes-Roth, et al, 83].

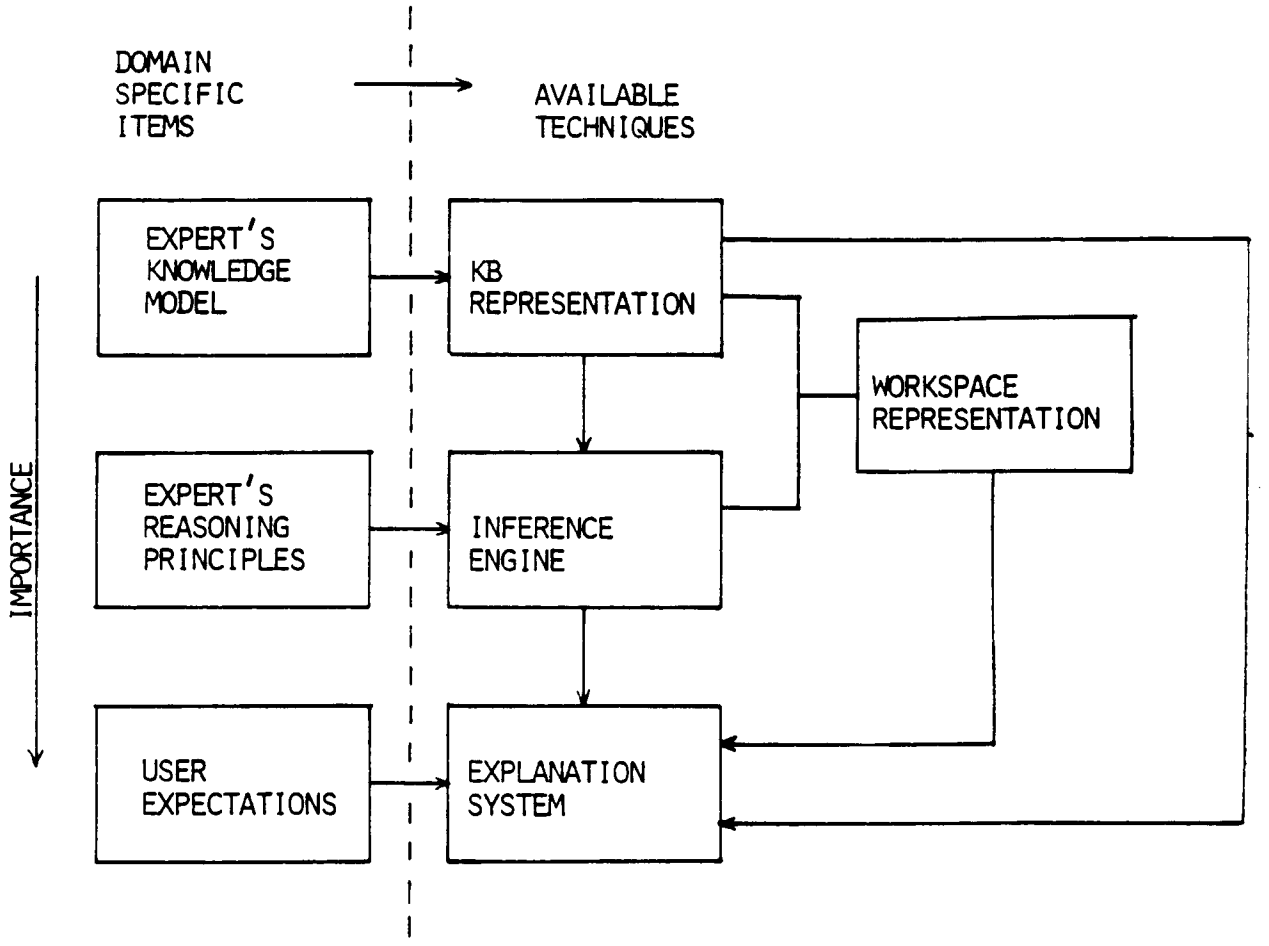


FIGURE 3-1. RESTRICTIONS ON CHOICES OF KBS METHODOLOGIES
BASED ON [BARNETT & BERSTEIN, '77]

3.1.3 Knowledge Representation Problems

In contrast to conventional data base systems, KBSs require a knowledge base with diverse kinds of knowledge - knowledge about objects, about processes, and hard-to-represent common sense knowledge about goals, motivation, causality, time, actions, etc. Attempts to represent this breadth of knowledge raise many questions [McCalla, 83]:

(1) How do we structure the explicit knowledge in a knowledge base?

(2) How do we encode rules for manipulating a knowledge base's explicit knowledge to infer knowledge contained implicitly within the knowledge base?

(3) When do we undertake and how do we control such inferences?

(4) How do we formally specify the semantics of a knowledge base?

(5) How do we deal with incomplete knowledge?

(6) How do we extract the knowledge of an expert to initially "stock" the knowledge base?

(7) How do we automatically acquire new knowledge as time goes on so that the knowledge base can be kept current?

In Section 3.2, some knowledge representation techniques are discussed, which answer some of the abovementioned problems.

3.1.4 Knowledge Representation Forms

Knowledge of a domain takes many forms through a KBS (Figure 3-2). A domain expert acquires knowledge through textbooks, journals, experience, etc. The expert's knowledge (or expertise) will be transformed to a knowledge acquisition (KA) facility in external form. The KA facility transforms the external representation into physical form (data structures, etc.) and stored in a knowledge base. This process is termed knowledge acquisition. It involves problem definition, implementation, refinement, and representation of facts and relations acquired from an expert. The KA process is discussed in detail in Section 3.5.4.

When an inference engine accesses the KB, the logical form (usually in the form of questions) is used at the interface. For example, during a problem solving activity, the IE could ask the KB whether a particular hypothesis is true or not.

From the IE, knowledge is transformed to advice or explanation when it reaches the user interface.

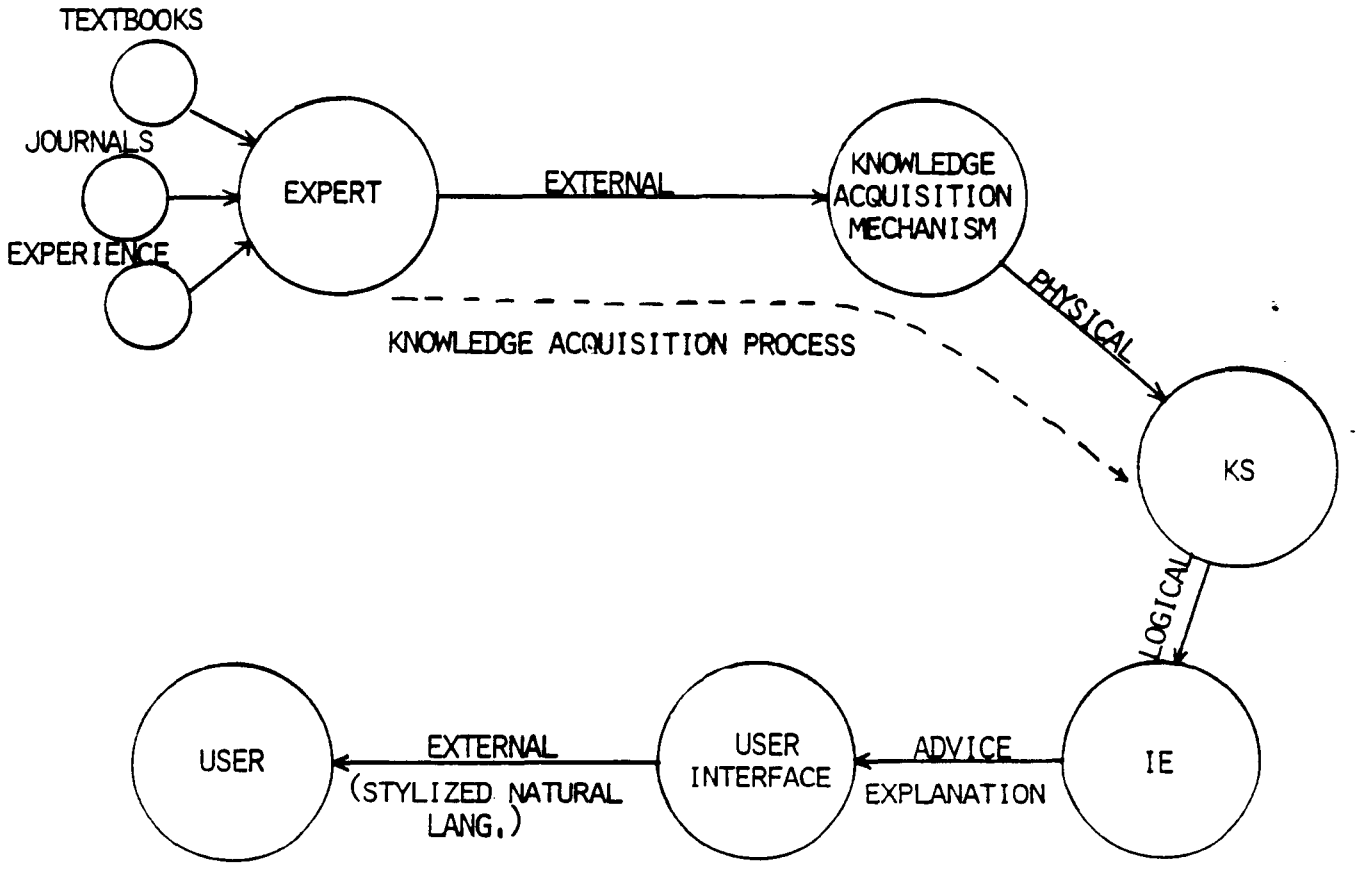


FIGURE 3-2. KNOWLEDGE REPRESENTATION FORMS
BASED ON [BARNETT & BERNSTEIN, '77]

Finally, knowledge is transformed back into external form (in stylized English) to the user.

Figure 3-2 summarizes the transformations of knowledge representations throughout a KBS.

3.1.5 Knowledge Representation Unit

Selection of a representation scheme for building a KBS is influenced by answers to questions of following type [Barr & Feigenbaum, 81]:

(1) In what detail are objects and events of the external world represented in a system?

(2) And how much of this detail is actually needed by the reasoning mechanism (or IE)?

The answer to these questions depends on the size of a knowledge chunk (also called grain size). A knowledge chunk is described as "a primitive unit in the knowledge representation, i.e., in a KB that contains the definitions of several interrelated terms, the definition of a single term is a 'chunk'" [Barnett & Bernstein, 77].

For example, in MYCIN, each rule of the type shown below is considered as a modular chunk of knowledge.

IF

1) The infection is primary bacteremia, and

2) The site of the culture is one of the sterilesites, and

3) The suspected portal of entry of the organism is the gastrointestinal tract

THEN

There is suggestive evidence (.7) that the identity of the organism is bacteriods.

There is no formal metric to define the "right" chunk size, yet it is a important consideration to KBS technology for three reasons:

(1) It determines the level at which the expert can instruct the system. If the correct chunk size is chosen, the expert could add and modify the knowledge base in a natural way. If, on the other hand, the chunk size is too big or too small, the expert is forced into an unnatural mode of expressing his knowledge.

(2) It influences the capability of an explanation facility, and user acceptance of a KBS, in part, depends on its explanation facility.

(3) It determines the kinds and efficiency of reasoning techniques to be used in the KBS. Larger chunk sizes

generally permit shorter lines of reasoning. For that reason, they are more likely to lead to a correct conclusion when inexact but plausible inference techniques are used [Barnett & Bernstein 77].

3.1.6 Credibility Factors

Expert systems are built to deal with real world problems in which reasoning is often judgemental and inexact or uncertain, i.e., axiomatic knowledge is not always available. There are two reasons for uncertainty :

(1) The expert who helps build the KBS may not be absolutely certain about a particular aspect of the problem domain.

(2) User may not be able to provide the necessary input data to the system or it may not be possible to obtain data within the time and other constraints.

In the former case, experts rate knowledge chunks as to their credibility or uncertainty when they enter them into the KB. In the latter case, relevant hypotheses or rules are combined with each other and with problem-specific parameters. The inference engine has the major responsibility in both cases.

Consider the following rule in MYCIN:

IF

1) The infection is primary-bacteremia, and

2) The site of the culture is one of the sterilesites, and the

3) The suspected portal of entry of the organ is the gastro-intestinal tract,

THEN

There is suggestive evidence (.7) that the identity of the organism is bacteroids.

The numbers used to indicate the strength of the rule (for example, .7 above) are called credibility factors (certainty factors, CFs, in MYCIN terminology).

The interpretation of CFs in the above example is that the evidence is strongly suggestive, (.7 out of 1) but not absolutely certain.

MYCIN evaluates its rules in three steps [Nau, 83] :

(1) The CF of a conjunction of several facts is taken to be the minimum of the CFs of the individual facts.

(2) The CF for the conclusion produced by a rule is the CF of its premise multiplied by the CF of the rule.

(3) The CF for a fact produced as the conclusion of one or more rules is the maximum of the CFs produced by the rules yielding that conclusion.

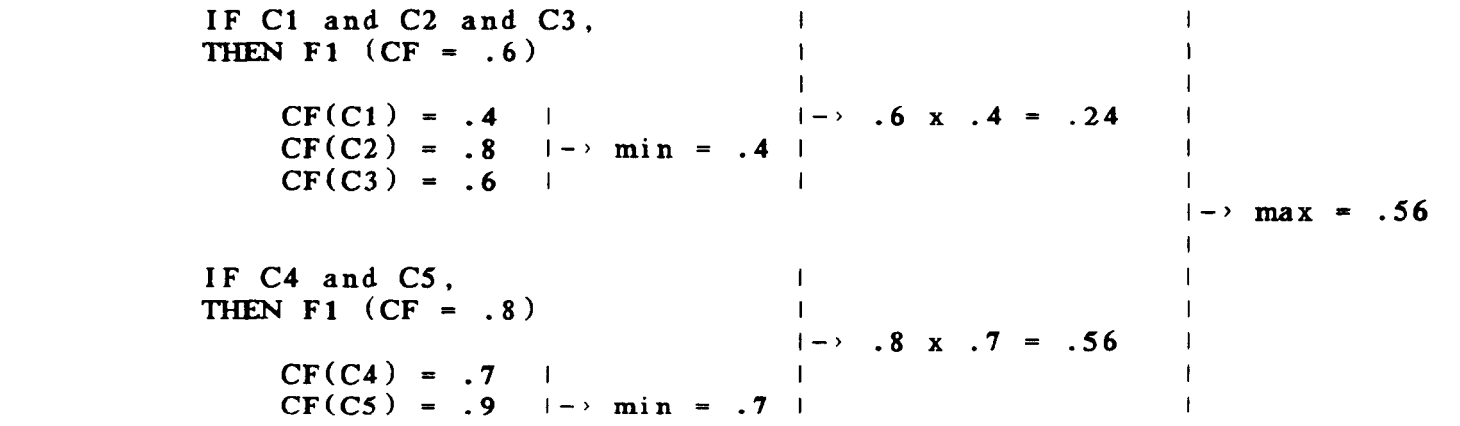
The following illustrates the above process.

Suppose MYCIN is trying to establish fact F1 and the only rules concluding anything about F1 are :

IF C1 and C2 and C3 THEN conclude F1 (CF = .6)

IF C4 and C5 THEN conclude F1 (CF = .8)

Further suppose that conditions C1, C2, C3, C4, and C5 are known with CFs .4, .8, .6, .7, .9, respectively. Then the following computation produces a CF of .56 for F1.



In the above example, we assumed that the conditions C1, C2, C3, C4, and C5 were established by other rules.

There are at least three other meanings or interpretations of credibility factors [Barnett & Bernstein, 77] :

(1) A Probability: the fraction of the time the chunk is true.

(2) Relevance: what is the probability that use of this chunk will ultimately lead to a completed chain of reasoning that solves the problem at hand?

(3) Acceptability: is this a preferred method or fact to workers in the field?

Because the mathematics for combining and evaluating each of the four interpretations is different, there should be an agreement between the knowledge engineer (who builds the KBS) and the expert (who instructs the system) as to the kind of credibility factors to be used.

A different approach, called "fuzzy logic", in dealing with uncertainity is described in [Zadeh, 75].

3.1.7 Procedural Versus Declarative Representation

In the area of AI, there had been a "battle" between proponents of procedural representation of knowledge (proceduralists) and advocates of declarative representation of knowledge (declarativists) much similar to the battle in the area of computer architecture between stack architecture advocates and register architecture advocates.

In the case of AI, at least, the issue is dissolved, rather than being resolved and one may argue that (1) there is no strictly formal difference in the power of the two - they are

both "universal" - and that (2) both are necessary [Barr & Feigenbaum, 81]. The major issue is management of complexity. KBSs seem to have done well in this aspect by selecting narrow and specific problem domains.

Declarativists argue that, using reasonably modular and independent knowledge chunks that are combined by a general purpose reasoning mechanism, a system can produce results that can be used for multiple purposes. The other qualities of declarative representation claimed by declarativists are: flexibility, economy, completeness, certainty, and modifiability.

Proceduralists, on the other hand, argue that some human knowledge (or intelligent behavior):

(1) Seems inherently non-modular.

(2) Is difficult to express as independent rules or facts.

(3) Has the ability to apply specialized rules to exploit situation-dependent relationships among knowledge chunks.

Hence a proceduralist believes that many ad hoc interrelationships should be made explicit and that procedures are the best way to do this [Barnett & Bernstein, 77]. The other qualities claimed by proceduralists are: directness, ease of coding, and understandab lity of the reasoning process itself.

The following example illustrates some of the issues involved.

A declarative representation of the statement, "All computer science (CMPS) majors at USL are smart" could be

For all x, [USLStudent(x) & CMPSMajor(x) --› Smart(x)]

A simple reasoning mechanism could use this single statement for many purposes. For instance, to answer the question, "Is Lin smart?", it would check to see whether Lin is a USL student and a CMPS Major. The answer is "yes". The same statement (or fact) could be used to infer that "Joe is not a CMPS Major" given the fact that "Joe is a stupid student". This example illustrates that an explicit representation of knowledge or a fact can be used for multiple purposes.

In a strictly procedural representation, the statement needs to be represented differently for each usage. Each would demand a specific form of the type "If you find a USL student, check to see whether he/she is a CMPS Major, and if so, assert he/she is clever".

An example to illustrate the advantages of procedural representation is provided below. The example is taken from [Kuipers, 75].

Consider a robot which manipulates a simple world such as a table top of toy blocks. This can be done most naturally by describing its manipulations as programs. The knowledge about

building stacks is in the form of a program to do it. Since we specify in detail just what part will be called when, we are free to build in assumptions about how different facts interrelate.

For example, we know that calling a program to lift a block will not cause any changes in the relative positions of other blocks (making the assumption that we will only call the lift program for unencumbered blocks). In a declarative representation, this fact must be stated in the form something equivalent to

> "If you lift a block X, and block Y is on block Z before you start, and if X is not Y and X is not Z and X is unencumbered, then Y is on Z when you are done".

This fact must be used each time we ask about Y and Z in order to check that the relation still holds. This knowledge is taken care of "automatically" in the procedural representation because we have control over when particular knowledge will be used, and deal explicitly with the interactions between the different operations.

3.2 Methods of Representing Knowledge Sources

3.2.1 Introduction

"Knowledge differs from information in that it is a property of the knower, interpreted by him through an internal

representation system, preparing him for action" [Kochen, 74].

This highlights the importance of efficient modes of representation. The underlying problem of understanding knowledge is the question of how to represent large amounts of knowledge in a fashion that permits their effective use and not that of finding some powerful techniques of implementing intelligent systems [Goldstein, 77].

The two major approaches are:

(1) Power-based strategy.

(2) Knowledge-based strategy.

In the first approach, we try to increase the computational power of the machine to be able to perform an efficient search and matching process. Many researchers have realized that this is not a constructive idea as these methods get overwhelmed by combinatorial explosion.

Instead, it would be useful to find better ways to express, recognize and use various forms of knowledge. A person is termed superior in intelligence because of his efficient and structured form of representing knowledge and associating it with different situations rather than the crude power called "thinking".

Having realized the importance of knowledge representation for efficient KBSs, we have to choose an appropriate form.

Different methods of representing knowledge are:

1. Finite state machines.
2. Programs.
3. Predicate calculus.
4. Production rules.
5. Semantic networks.
6. Frames.

Feigenbaum [Feigenbaum, 81] has very beautifully stated that an encyclopedia cannot be termed knowledgeable (or containing knowledge) unless one knows how to extract useful information out of it. The above mentioned methods are supposed to achieve the same goal. The intelligence of any KBS will depend on how efficiently these methods will help programs to extract and interpret knowledge contained in the knowledge base. The representations are broadly classified into

(1) Declarative
(2) Procedural

The names themselves suggest their meaning (see Section 3.1.7). In the first one, we "declare" bits of knowledge which will be used by the system to "deduce" certain results. It is highly mechanical and helps to derive concrete results. Its main disadvantage is that it may get drowned in a combinatorial explosion created by itself. The other method involves procedures for accomplishing certain tasks. Thus, depending on

the set of rules followed, certain conclusions can be derived from the procedures. The problem lies in the fact that the procedures might be unable to conclude for many instances.

Thus, if we could overcome the limitation of declarative methods by combining them with procedural methods, it might be possible to evolve a more efficient method of knowledge representation. This way we could have the advantage of ease in modification provided by declarative representation along with the directedness of procedural representation.

It has been very rightly said by Newell [Newell, 82] that,

Representation = Knowledge + Access.

This means that we should represent knowledge such that we have a system to provide access to it, such that it helps us to select a certain action for reaching our goal. The representation is the structure which realizes knowledge and reduces it to the next lower level.

At times, it has been found advantageous to combine knowledge representations of different types [Aikins, 83]. It should also be possible to use the same knowledge base for multiple uses. Thus, the topic of knowledge representation demands thorough understanding for developing efficient intelligent systems.

Knowledge representation forms the heart of KBSs (or Expert Systems). The strength of the system lies in the depth as well as the breadth of knowledge represented in the system. Thus, it

is quite desirable at the time of designing a new system to decide on the knowledge representation technique to be adopted. There are a few generalized techniques of knowledge representation which could be used. Many systems designers prefer to design their own knowledge representation technique which might be a slight modification of one of the major representation techniques.

It is virtually impossible to get information on all the knowledge representation techniques. As quite a few of them are application dependent, they may not be useful to other systems. Thus, this discussion will concentrate on a few generalized knowledge representation techniques. Wherever possible, examples are provided to help the reader in understanding these techniques.

According to Feigenbaum [Feigenbaum, 81], at present, there is no theory of knowledge representation. We are also not in a position to prove that one system represents human memory better than any other. The objective of this section is to highlight why certain systems work efficiently for certain knowledge representations.

3.2.2 Finite-State Machine

3.2.2.1 Introduction

A finite state machine (FSM) is a knowledge representation technique of procedural type.

The FSM, as the name suggests, is a collection of a finite number of states. Each state specifies actions (or computations) that should be taken to reach the next state. There are two special states in a FSM. A start state is the initial state and an end state is where action or computation terminates.

FSMs are widely used in planning strategies, in designing digital electrical circuits (adders, flip-flops, multipliers, etc.), and to represent grammars [Woods, 73].

3.2.2.2 Example 1

A simple example of a finite state machine is a lamp with a pull-chain (Figure 3-3). Pulling the chain turns the light on if it is off and off if it is on.

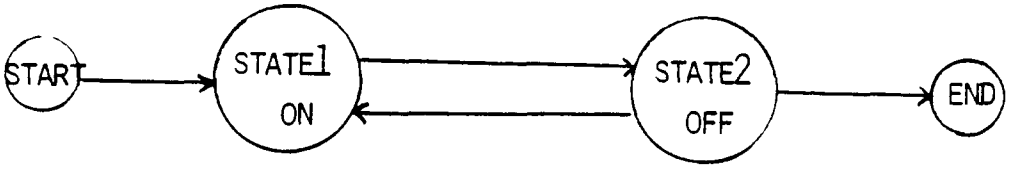


Figure 3-3. Finite State Machine Representation of a Lamp with a Pull Chain.

Figure 3-3 is a state-transition diagram of a pull-chain lamp. Circles represent states. Transitions are represented by arcs (or arrows). The actions (or inputs) are represented on the arcs and reactions (or outputs) are on the right side of the input separated with a slash. State S1 is the "lights on" state and, by pulling the chain, a transition is made to state S2, "lights off". Likewise, from state S2 ("off"), by pulling the chain, transition is made to state S1 ("on").

The power, size, and reversibility (the ability to reach an initial state from a final state) of a FSM depend on the following four issues [Barnett & Bernstein, 77] :

(1) The set of allowable computations in a state.
(2) The set of decision rules (or predicates) that take a FSM from one state to another state.
(3) Parameterization.
(4) The control mechanism.

3.2.2.3 Example 2

The Figure 3-4 illustrates some of the issues involved in a finite state machine representation of knowledge. The circles represent states. Arrows (or arcs) represent transitions. Actions are represented inside the circles. Decision rules or predicates are represented on the arcs. Decision rules must be satisfied in order to go from one state to another.

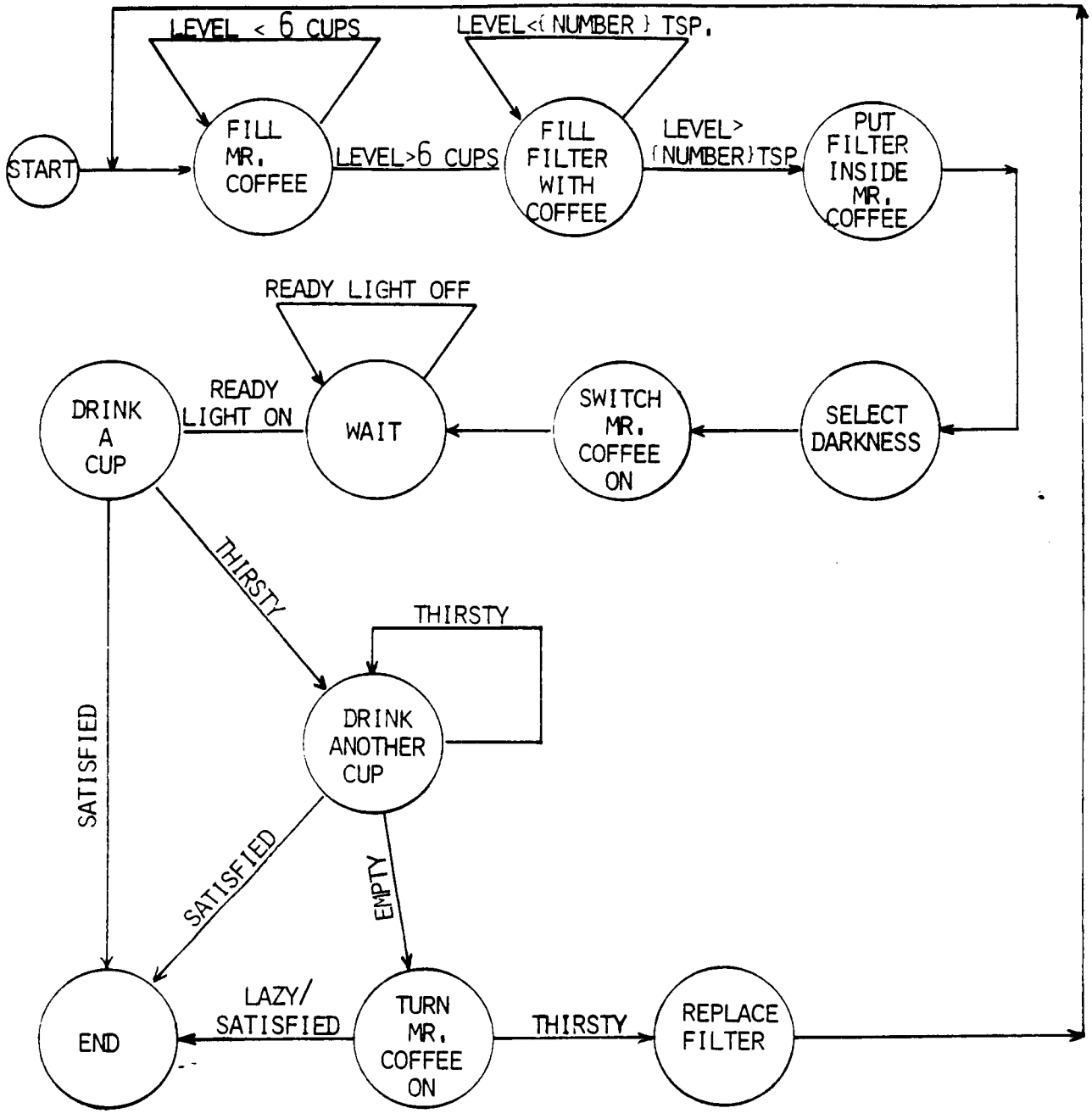


FIGURE 3-4. FINITE STATE REPRESENTATION OF A PLAN TO MAKE AND DRINK COFFEE USING "MR. COFFEE"

For example, in Figure 3-4, the state marked "Wait" has two arcs leaving it. One is labeled "Ready Light Off". FSM will be in this "Wait" state - then a FSM is said to be blocked - until the ready light turns on. When this happens, the FSM goes to its next state, "Drink a Cup" in our example.

We can also use parameters in a FSM. In our example, the number of table spoons of coffee that are to be used in filling the filter is passed as an argument (NUMBER) on the arcs leaving the state "Fill Filter with Coffee".

3.2.2.4 Control Mechanism

The power of a FSM, as mentioned earlier, also depends on its control mechanism. There are two types of control: deterministic and non-deterministic.

(a) Deterministic

In a deterministic FSM, one arc predicate controls the transition from one state to another. This is accomplished either by requiring that at most one arc predicate be true, or by having a rule that selects one arc out of the set that qualifies. In our example, the state "Drink Another Cup" has three arcs leaving it: "Thirsty", "Empty", and "Satisfied". One cannot drink coffee from an empty MR. COFFEE even if he is thirsty. So there should be a selection rule which gives priority for the arc "Empty".

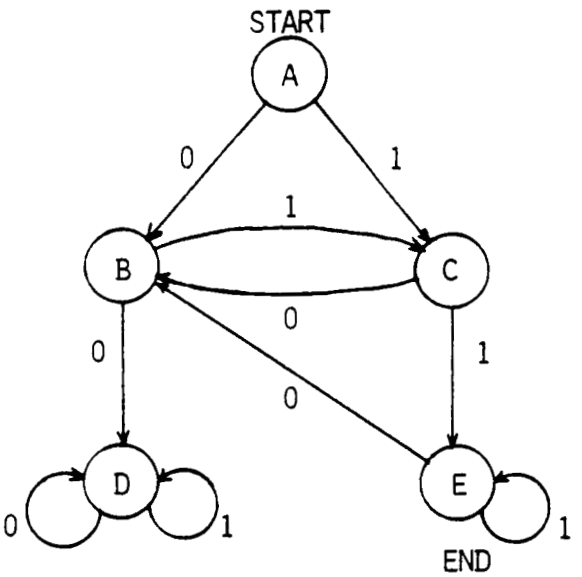
(b) Non-Deterministic

In a non-deterministic FSM, it is possible for several different arcs leaving the same state to be satisfied simultaneously. Thus, in a non-deterministic FSM, the next state is not completely determined by the current state and its input. Instead, a set of next possible states is to be determined. If any arc reaches the end state, the FSM will terminate normally.

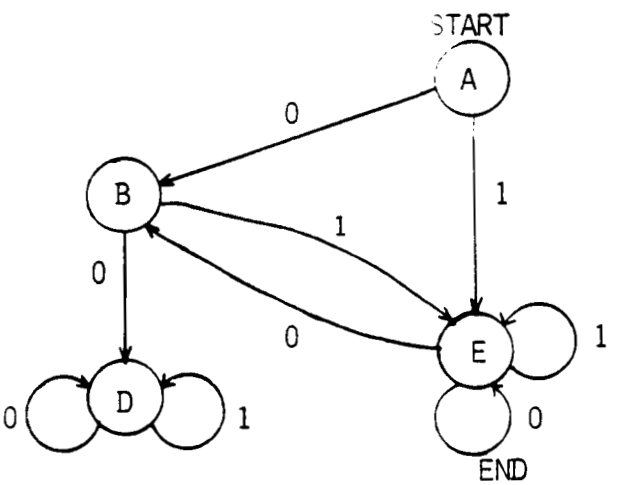
An example which illustrates the differences between deterministic FSM and non-deterministic FSM is presented below.

Figure 3-5 shows both a deterministic and non-deterministic FSM that recongnize symbol strings that start with one or more "01" and ends with two consecutive 1s and does not contain two consecutive 0s.

In Figure 3-5 circles represent states and the letters inside the circles represent the state names. Thus "A" is the start state and "E" is the final state. Arcs represent state transitions and symbols on the arc represent the inputs (the symbol that is scanned) that cause those transitions.



A. DETERMINISTIC



B. NON DETERMINISTIC

FIGURE 3-5. FINITE STATE RECOGNIZERS FOR $\{0,1\}^*$ ENDING WITH 2 CONSECUTIVE 1S AND DOES NOT CONTAIN TWO CONSECUTIVE 0S.

(a) Deterministic FSM

Suppose the input string is "010111". Starting in the state "A", the successive states into which it is thereafter driven are (in order) B, C, B, C, E, E. Since E is the final state, the deterministic FSM correctly recognizes the input string "010111". If the input is "10011", beginning in state "A", the successive states into which it is thereafter driven are (in order) C, B, D, D, D. Since D is not a final state, the deterministic FSM (correctly) fails to recognize "10011".

(b) Non-Deterministic FSM

Again suppose the input string is "010111". Starting in the state "A", one possible sequence of states into which it can be thereafter driven are (in order) B, E, B, E, E, E. Since E is the final state, it correctly recognizes the input string "010111". Another possible sequence of states is B, E, E, E, E, E, which correctly recognizes the input string.

Now suppose the input is "10011". Starting from state A, one possible sequence of states it thereafter driven is E, E, E, E, E. Since E is a final state, the non-deterministic FSM incorrectly recognizes the input string "10011". Another possible sequence of states is E, B, D, D. Since D is not a final state, the non-deterministic FSM (correctly) does not recognize the input string.

In the above example, the deterministic FSM has one more

state than the non-deterministic FSM. There are some cases where this factor makes a critical difference in implementing non-deterministic control over deterministic control. However, interpretation of a non-deterministic FSM (by an inference engine in a KBS) is more complex.

This section on FSM is concluded by discussing the desirable and undesirable characteristics of a FSM. The discussion is based on [Barnett & Bernstein, 77].

3.2.2.5 Characteristics of FSM

The desirable characteristics are:

(1) The ability to easily implement nondeterministic control.

(2) The ability to represent and model plans of action for which "procedural" execution inside a computer is meaningless.

(3) Reversibility, i.e., an FSM may be examined to answer such questions as what needs to occur to allow it to end up in a particular state.

(4) New plans of action may be constructed dynamically because an FSM representation is easily manipulated.

(5) Many disciplines, both scientific and nonscientific, represent part of their published expert knowledge in

a form similar to that of an FSM.

The undesirable characteristics of FSMs are:

(1) The loss of efficiency compared to compiled procedures.

(2) The enforcement of low-level uniformity in the representation, which can make the FSM hard to understand (in a sense, FSMs are better at representing strategies than tactics).

(3) The external format of an FSM representation can lose clarity unless there is a graphic medium available for computer input and display.

3.2.3 Using Programs to Represent Knowledge

3.2.3.1 An Example

Procedural knowledge can be represented by programs. Figure 3-6 depicts a program representation of knowledge necessary to adjust the volume of a stereo set. The example has two arguments: a human agent who will perform the task, and the desired volume of the stereo set. Much world knowledge (common sense knowledge) is embedded in this program. For example,

(1) Stereos are in houses, cars, etc.

(2) You need to be close to the stereo to control the volume

(3) Turning the knob clockwise increases the volume (rightmost - highest or loudest) and turning counter clockwise reduces the volume (leftmost - lowest) and volume can be adjusted by adjusting the knob.

(4) Before the volume can be adjusted, the stereo set must be switched on.

(5) Relative values of loudness such as high, low, medium, etc. are used and compared.

Besides this world knowledge, the program contains knowledge about itself - for example,

(1) The program will not go into an infinite loop while trying to adjust the volume, because only approximate equality is necessary to terminate.

(2) Program "MOVE" will effectively move the agent to the desired location, room, in our example.

(3) Program "ROTATE-KNOB" expects the agent to be in proximity of the stereo set.

```
PROCEDURE  ADJUST_THE_STEREO (AGENT human, DESIRED_VOLUME volume)

      MOVE (AGENT, "room");
      IF   DESIRED_VOLUME = "High" or "Loud"
      THEN Knob_direction <- "Right";
      ELSE Knob_direction <- "Left";
      ROTATE_KNOB (AGENT, "Right", "Full_turn");
      IF   Knob_direction = "Left"
      THEN DO:
           x <- "Half_turn";
           WHILE (CURRENT_VOLUME > DESIRED_VOLUME) DO
                 IF   CURRENT_VOLUME > DESIRED_VOLUME
                 THEN ROTATE_KNOB (AGENT, "Left", x);
                 ELSE ROTATE_KNOB (AGENT, "Right", x);
                 x <- x/2;
           END  /* WHILE */;
      END  /* THEN */;

END /* ADJUST_THE_STEREO */;
```

Figure 3-6. Procedural Knowledge Example

The advantage of the program representation is that all of the knowledge is represented in a natural manner. The disadvantages become apparent if one tries to extend this example to stereo sets where sliding a indicator up and down adjusts the volume.

When programs are used to represent knowledge, two options are available : invocation methods and control structures.

3.2.3.2 Invocation Methods

The four methods of program invocation are: direct, procedural attachment, demon, and pattern directed.

(a) Direct

Direct invocation occurs when the user (using program) knows precisely which program is to be used and some identification (for example, name) is used to reference that program through a mechanism such as a subroutine call.

(b) Procedural Attachment

The basic concept of procedural attachment (PA) is that most knowledge should be expressed declaratively (as data structures or items) and should permit optional association of programs with the knowledge chunks and/or the data items within the chunks. Whenever these knowledge chunks are referenced, the program(s) associated with them will be executed. The invoker of the program may be unaware both what program is invoked and what functions the invoked program is to perform. Usually, only the program that makes the attachment has that knowledge.

(c) Demons

A demon is like an interrupt handler in an operating system. They perform no action unless and until a specific situation is encountered. They allow knowledge that pertains to highly

specialized or unusual situations to be left out of the main stream, making programs more readable and easier to organize.

(d) Pattern-Directed

In a system using the pattern directed (also known as goal-directed) method, each program is named by a pattern that describes the kind of tasks it performs.

An example of a pattern for the "MOVE" goal (Figure 3-6) is MOVE(human, object). This states that the program can plan the sequence of actions necessary to move a human into proximity to an object. Another program in the same system could have a pattern such as MOVE(object1, object2). To move either object1 or object2, an external agent may be required. Thus, the second program performs a different task from that of the first program.

3.2.3.3 Control Structures

Control structures in programs can be sequential or parallel or non-deterministic.

(a) Sequential

In a sequential method, the program itself explicitly makes the choice of what to do next.

(b) Parallel

In a parallel method, many subprograms can operate

simultaneously and programs themselves are responsible for synchronization mechanisms.

(c) Non-Deterministic

In a non-deterministic method, each program, when operating, will have the same environment, and many branches will be followed during execution.

3.2.3.4 Advantages and Disadvantages

See Section 3.1.7, "Procedural vs. Declarative Representation"

3.2.4 Predicate Calculus

3.2.4.1 Introduction

The predicate calculus is a formal notation system (i.e., formal language) that can be used to represent knowledge in AI systems.

In the next section, a predicate calculus definition is presented. In Section 3.2.4.4, an example to illustrate the concepts is presented and in Section 3.2.4.6, the advantages and disadvantages of using predicate calculus to represent knowledge in AI systems will be discussed. The definition and discussion of the predicate calculus are based on an excellent book by Nilsson [Nilsson, 71], and [Barnett & Bernstein, 77] (p. 76-88).

3.2.4.2 Predicate Calculus Definition

There are three parts to the definition of PC.

(a) Syntax specification - the grammar that defines legal expressions in the language.

(b) Semantic specification - the rules that relate the symbols in the language to objects in the domain.

(c) Legal operations - rules of inference that create legal expressions from other legal expressions.

The syntactically legal expressions in the predicate calculus are called "Well-Formed Formulae" (WFF). Through the semantic specification rules, a WFF makes an assertion about the domain. The WFF are said to have the value T or F, depending on whether the assertions are true or false on the domain. The legal operations are constrained in such a way that the value (T or F) of a WFF output by a transformation can be directly determined from the values of the WFFs input to the transformation.

(a) Syntax

The syntax specification of the first-order predicate calculus (higher orders will be discussed later) has two parts:

(1) The specification of an alphabet of symbols.

(2) The method by which legal expressions are constructed from these symbols.

The alphabet consists of the following set of symbols:

(1) Punctuation marks: , ( )

(2) Logical symbols: - => v

(3) Quantifier symbols: $\forall$ $\exists$ (The symbol $\forall$, is called the universal quantifier and is read for all; the symbol $\exists$ is called the existential quantifier and is read as there exists.)

(4) n-adic function letters: $f^{n}(i)$ $(i >= 1, n >= 0)$
(The $f^{0}(i)$ are called constant letters.

(5) n-adic predicate letters: $p^{n}(i)$ $(i >= 1, n >= 0)$
(The $p^{0}(i)$ are called proposition letters.)

(6) Variables: x(i)

From these symbols, the definition of a WFF can be recursively expressed:

1. Terms

   a. Each constant letter is a term.

   b. Each variable letter is a term.

c. If $f^n(i)$ is a function letter and t(1) t(2) .. t(n) (n >= 1) are terms, then $f^n(i)$ (t(1), t(2), .. t(n)) is a term.

d. No other expressions are terms.

2. Atomic formulae (Domain-specific Boolean-valued expressions)

a. The propositional letters are atomic formulae.

b. If t(1) t(2) .. t(n) (n >= 1) are terms and $p^n(i)$ is a predicate letter, the expression $p^n(i)$ (t(1), t(2) .. t(n)) is an atomic formula.

c. No other expression is an atomic formula.

3. WFFs

a. An atomic formula is a WFF.

b. If A and B are WFFs, then so are

i (~A) (Read as not A)

ii (A => B) (Read as A implies B)

iii (A V B) (Read A or B (or both))

iv (A B) (Read as A and B)

c. If A is a WFF and x is a variable, then the following are WFFs:

i ($\forall$ x)A (Read as for all x, A)

ii ($\exists$ x)A (Read as, there exists x such that A)

d. No other expressions are WFF.

The parentheses shown in 3b and 3c are usually omitted where no confusion will result. Some of WFFs, using abbreviated notation, are:

$-P(a, g(a,b,a))$

$P(a,b) \Rightarrow (\exists y)(\exists x)(Q(a,y) \vee S(x,y,a))$

$(LESS(a,b) \ (b,c)) \Rightarrow LESS(a,c)$

Some examples of expressions that are not WFFs are:

$-f(a)$

$h(P(a))$

$Q(f(a), (P(b) \Rightarrow Q(c)))$

(b) Semantics

The semantic specification rules for the predicate calculus give a "meaning" to the WFFs by making a correspondance between symbols in the calculus and objects in the domain. The domain, D, is a nonempty set of objects. The necessary correspondances are [Barnett & Bernstein, 77]:

(1) Associated with every constant symbol in the WFF is some particular element of D.

(2) Associated with every function letter in the WFF is an n-adic function over (and into) D.

(3) Associated with every predicate letter in the WFF is

some particular n-place relation among the elements of D. (A relation may be considered as a function whose only values are T and F.)

(c) Interpretation (or Inference)

The specification of domain and the above semantic associations constitute an interpretation or a model of the WFFs. Given a WFF and an interpretation, we can assign a value, T or F, to each atomic formula in the WFF. These values can be used in turn to assign a value, T or F, to the entire WFF. The process by which a value is assigned to an atomic formula is straightforward: If the terms of the predicate letter correspond to elements of D that satisfy the associated relation, the value of the atomic formula is T; otherwise, the value is F. For example, consider the atomic formula:

P(a, f(b,c))

and the interpretation

D is the set of integers

a is the integer 2

b is the integer 4

c is the integer 6

f is the (two-argument) addition function

P is the relation greater-than

With this interpretation, the above atomic formula asserts

that "2 is greater than the sum of 4 and 6". In this case, the assertion is false and P(a, f(b,c)) has the value F. If the interpretation is changed so that a is the integer 11, then the value is T.

The method of assigning a value to an atomic formula containing variables is not so simple. For example, the atomic formula:

$(\forall x)\ P(f(x,a),\ x)$

with the interpretation

D is the set of integers

a is the integer 1

f is the (two-argument) addition function

P is the relation greater-than

makes the assertion, "for all x in D (x any integer), x plus one is greater than x". Hence, the atomic formula has a value only under the "influence" of the quantifier. When more than one quantifier is used, then the operation of each may depend upon those further to the left. Let the interpretation be

D is the set of integers

P is the relation greater-than

Then, the WFF,

$(\forall x)\ (\exists y)\ P(y,x)$

C-2

asserts that for all x (integer) there exists a y (integer), which may depend upon the chosen x, such that y is greater than x. The value of this WFF is T. However, the WFF

$$(\exists y)\ (\forall x)\ P(y,x)$$

asserts that there exists a y (integer) such that y is greater than any (integer) x. The value of this WFF is F.

The values of WFFs composed using logical symbols are derived by a set of rules that are independent of the interpretation. If X is any WFF, then (~X) has the value T when X has the value F, and (~X) has the value F when X has the value T. Table 3-2 shows how the values of WFFs composed by the other logical connectives are determined from the values of the WFFs X(1) and X(2).

Given these definitions of the logical and quantifier symbols, it is easy to show that the symbols V, $\wedge$, and $\exists$ are redundant because they can be expressed in terms of the symbols ~, => and

$$X(1)\ \ X(2) = \sim(X(1) \Rightarrow \sim X(2))$$

$$X(1)\ v\ X(2) = (\sim X(1) \Rightarrow X(2)$$

Table 3-2. DEFINITION OF THE LOGICAL CONNECTIVES

X1	X2	X1 V X2	X1 X2	X1 -› X2
T	T	T	T	T
F	T	T	F	T
T	F	T	F	F
F	F	F	F	T

3.2.4.3 Some Definitions

Several terms are used to describe properties of WFFs and the calculus itself:

Valid. A WFF that has the value of T for all interpretations is called valid.

Decidable. A calculus is called decidable if there exists a general method for determining, for any WFF in that calculus, whether it is valid.

Undecidable. If a calculus is not decidable, then it is undecidable.

Satisfy. If the same interpretation makes each WFF in a set of WFFs have the value T, then this interpretation is said to satisfy the set of WFFs.

Unsatisfiable. If no interpretation exists such that each WFF simultaneously has the value T, then the set of WFFs is said to be unsatisfiable.

Prove. To prove W given S means to show that W logically follows from S.

Propositional Calculus. If the use of quantifiers and variables is prohibited, the result is called the propositional calculus, a decidable subset of the first-order predicate calculus.

Second-order Calculus. A second-order predicate calculus comes about by allowing quantification of propositional letters in addition to the quantifications allowed in the first-order theory.

Omega-order Calculus. The second-order calculus can be extended by allowing quantification of the higher-order predicate letters. Such a calculus is called omega ordered predicate calculus.

The predicate calculus provides a natural way of expressing declarative knowledge. A knowledge source is a collection of WFFs and the semantic rules that relate them to the domain of application. The included WFFs all have the value T and are called axioms. The semantic rules are usually straightforward and implicit, i.e., the abbreviated names used for the f(i) and p(i) are chosen in such a way that the correspondance to the domain is intuitive.

3.2.4.4 An Example

The following example illustrates many of the concepts involved in predicate calculus. This example (Figure 3-7) is taken from [Klahr, 78]. There are four axioms:

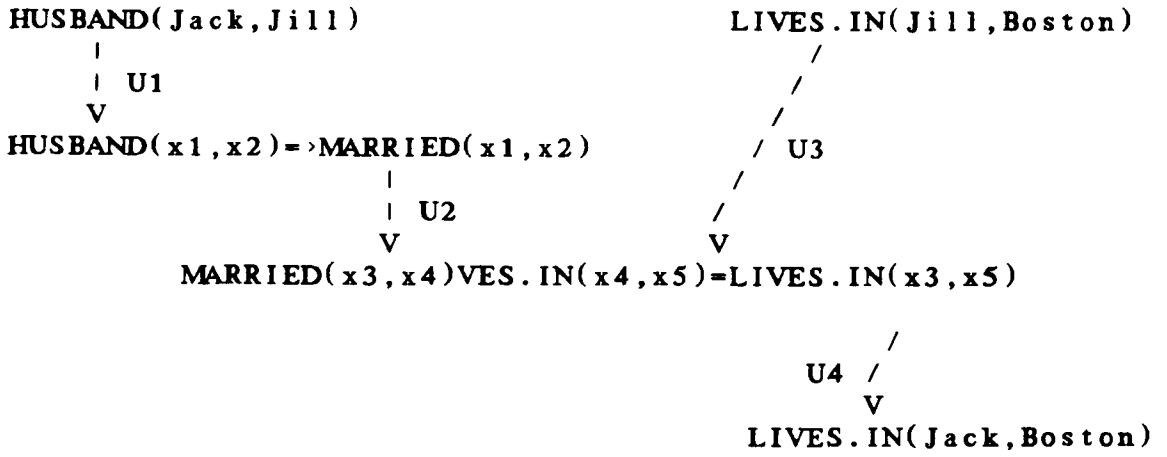
(1) Jack is the husband of Jill.

(2) Jill lives in Boston.

(3) If x1 is the husband of x2, then x1 and x2 are married.

(4) A married couple lives in the same place.

```
AXIOMS: (1) HUSBAND(Jack,Jill)
        (2) LIVES.IN(Jill,Boston)
        (3) (∀x1)(∀x2)(HUSBAND(x1,x2)⇒MARRIED(x1,x2))
        (4) ~(∀x3)(∀x4)(∀x5)((MARRIED(x3,x4)VES.IN(x4,x5))⇒
                    LIVES.IN(x3,x5))
```




```
                    U1   U2   U4
Variable chains: Jack-->x1-->x3-->Jack
                    U1   U2
                 Jill-->x2-->x4

                       U3
                 Jill-------+

                      U3   U4
                 Boston-->x5-->Boston
```

Theorem: LIVES.IN(Jack,Boston)

Figure 3-7. Proof that Jack Lives in Boston
[Barnett & Bernstein, 77]

The assertion derived is "Jack lives in Boston". The proof is shown schematically with the reasoning chain depicted by the single arrows. Thus, the proof consists of the above axioms as steps(1) through (4) followed by:

(5) Jack is married to Jill - because of (1) and (3).

(6) Jack lives in Boston - because of (2), (4), and (5).

When passing along the arrows, an association is established between the variables and/or the terms on each side of the arrow. For example, along the arrow labeled U1, x1, and x2 are respectively associated with Jack and Jill, and along the arrow labeled U2, x1, and x2 are respectively associated with x3 and x4. Each such association is called a unification. The set of all such unifications are summarized, under the heading "Variable chains", at the bottom of the Figure 3-7 There are three chains in the example: (Jack x1 x3), (Jill x2 x4), and (Boston x5). The chains are formed as equivalence classes of terms and variables so that each variable is in one and only one chain, no variable in one chain unifies with a variable in another chain, if the chain contains more than one element then each element unifies with at least one other element in the chain, and the number of chains is maximal.

In order to prove an assertion three rules must be followed:

(1) At most one term can occur in an equivalence class - all variables in the class then have this value.

(2) If no terms occur in a class, then there must exist an object in the domain such that all variables in the chain may legally assume that value.

(3) Either rule (1) or (2) must apply simultaneously to every chain.

The example shows a method of determining a value (in this case T) of the assertion, "Jack lives in Boston." This raises the natural question of how to deal with the problem, "Where does Jack live?" The method described in [Nilsson, 71] for solving this kind of problem is based on the resolution technique for generating proofs in the first-order predicate calculus. The method consists of two parts:

(1) Use resolution to generate a proof for a related problem - for example, $(\forall x)$ LIVES.AT(Jack,x); and

(2) Use the generated proof to find an appropriate answer to the problem - in this case, x = Boston.

3.2.4.5 Characteristics of Predicate Calculus

One of the features of the predicate calculus is the ability to derive new facts and beliefs using some existing WFFs. This is a good idea, but it falls short as a means of representing knowledge in KBSs and other AI applications. One of the difficulties is that it is not enough simply to have the "facts

at hand"; one must know how to use them. Consider for example, the inference rule OR-introduction

A => A V B

OR-introduction captures the idea that we can infer "A or B" either by proving A or by proving B. Given constants D, E, and F, we an use this rule to infer

D V E

D V F

as well as wonders as

D V D

D V E V E

D V E V D V E

D V E V E V E V E V E

and so on without limit.

This example (based on [Hayes-Roth, et al, 83]) shows that the unguided application of inference rules can be explosive. The inferences are perfectly correct; they are just not particularly interesting. And this contributes to what is called combinatorial explosion in large search problems (see Section 3.3.4).

Much work has been directed toward controlling combinatorial explosion. For example, some mechanical theorem-proving techniques avoid nonsense applications of OR-introduction. Methods that use many rules of inference need to incorporate knowledge to control their use [Hayes-Roth, et al, 83]. Some

alternative but equally troublesome methods are suggested (see [Nilsson, 80]) for example, resolution and resolution strategies.

Another characteristic of predicate calculus representations is demonstrated by example of Figure 3-7 namely, there are two broad categories of axioms [Barnett & Bernstein, 77]:

(1) First, there are specific facts such as "Jack is Jill's husband" or "Jill lives in Boston".

(2) Second, there are general assertions such as "Married couples live at the same place." In any actual application domain, the number of facts will be overwhelming. The result is impractically slow proof procedures or the use of different methods, in the inference engine, to handle facts and general knowledge. More detailed discussion on this problem can be found in [Kalhr, 78].

3.2.4.6 Advantages and Disadvantages of Predicate Calculus

Advantages:

(1) Predicate calculi are the best theoretically understood and among the oldest techniques used for representing knowledge in a computer.

(2) Predicate calculus is modular and reversible.

Disadvantages:

(1) Representing procedural knowledge in the predicate calculus is difficult.

(2) In predicate calculus, the entire set of axioms must be consistent. Thus, it makes it impossible to include heuristic and possibly contradictory rules of thumb and other sorts of expert knowledge in the knowledge base.

3.2.4.7 Systems That Use Predicate Calculus

Some systems that use predicate calculus languages to represent knowledge:

- QA3 [Green, 69], a general-purpose, question-answering system that solved simple problems in a number of domains.

- STRIPS, the Stanford Research Institute Problem Solver, is designed to solve planning problems faced by a robot in rearranging objects and navigating in a cluttered environment [Fikes, 72].

- FOL [Filman & Weyhrauch, 76] is a very flexible proof checker for proofs stated in first-order predicate calculus.

3.2.5 Production Rules as a Representation of Knowledge

3.2.5.1 Introduction

Many of the highly successful KBSs use production rules as the representation of knowledge in a knowledge base.

A production rule is a specification of conditional action and consists of a left hand side (LHS) (also called condition or antecedent), which describes a situation, and a right hand side (RHS) (also called action or consequence), which describes something that may legally be done in a situation described by the LHS [Barnett & Bernstein, 77].

For example, in "If you are outdoors and it is raining, then open umbrella", the conditions are (1) being outdoors, and (2) rain. The action is to open an umbrella.

3.2.5.2 Production System Types

There are (at least) three types of application areas where production rules are used as a knowledge representation mechanism [Davis & King, 77].

(a) Psychological Modeling

The attempts to simulate (or mimick) human performance (behavior) on simple tasks are aimed at creation of programs which embody a theory for that behavior. Using a minimum set of competent production rules, some psychological modeling

experiments (EPAM, [Barr & Feigenbaum, 81], for example) were able to reproduce the behavior. Here the "behavior" is meant to include all aspects of human shortcomings or successes which may arise out of (and hence may be clues to) the "architecture" of the human cognitive system [Davis & King, 77]. Some of these shortcomings like oscillation and forgetting may be considered as "mistakes" for a system intended for high performance, but are important in a system meant to model human learning behavior [Feigenbaum, 63].

A system with the above described behavior is described in [Newell & Simon, 72].

(b) Formal Language Theory

In some formal language theories, production rules have been used to write grammars for formal languages [Floyd, 61], [Evans, 64]. The important characteristic of these theories is that they use non-determinism for control structure and rule selection.

(c) Knowledge Based Systems

These systems use production rules as a representation of knowledge about a task or domain and attempt to build a program which displays competent behavior in that domain. In these (expert) systems, there is no explicit attempt to "simulate" a specialist's problem solving behavior; however, the system derives power from integrating the same heuristic knowledge

experts use, with the same informal style of reasoning [Buchanan & Duda, 83].

The example and the rest of the discussion in this section is oriented towards this category.

3.2.5.3 Production System Components

A production system consists of three parts [Barr & Feigenbaum, 81]:

(a) A rule base - a collection of production rules.

(b) A workspace - a buffer like data structure.

(c) An interpreter or control mechanism - which controls the system activity.

(a) Production Rules

Production rules are represented by some agreed upon syntax. A set of primitives and symbols (that correspond to objects and functions in the domain) are used to construct LHS and RHS of production rules.

(b) Workspace

Workspace (sometimes called context, or data base or short term memory (STM) buffer) is the focus of attention of production rules. It contains the total description of the system's current state or situation. The LHS of a rule is matched against the contents of the workspace. If there is a match, then RHS is

executed ("fired") and RHS action modifies the workspace. Then a production rule is said to be applied.

(c) Interpreter (or control mechanism)

In a production system, the interpreter has three tasks:

(1) Matching or building a Conflict-Set - the set of all production rules whose LHSs are satisfied. If the conflict set is empty, then processing is terminated.

(2) Conflict-Resolution - if the conflict set is not empty, then one member of the conflict set is selected.

(3) Action or Execution - the RHS of the above selected production rule is executed.

The entire cycle is repeated until the termination condition is reached.

3.2.5.4 Conflict Resolution Strategies

Several conflict resolution strategies have been used or proposed. Among them are [Barnett & Bernstein, 77]:

(a) Rule Order: There is a complete ordering of all production rules. The rule in the conflict set that is highest in ordering is chosen.

(b) Rule Precedence: A precedence network determines an ordering.

(c) Generality Order: The most specific rule is chosen.

(d) Data Order: Elements of the workspace are ordered. The rule chosen is the one whose LHS references the highest-ranking workspace element(s).

(e) Regency Order: Execute the rule in the conflict set that was most (least) recently executed, or the rule in the conflict set whose LHS references the most (least) recently referenced element(s).

(f) Non-Deterministic: Execute every rule in the conflict set as if it were the only member. Computation stops when any path terminates.

3.2.5.5 Example 1

The following example (a slight modification of [Barr & Feigenbaum, 81] page 191] illustrates some of the basics of production system.

Consider a production system (PS) that might be used to identify a food item, given a few hints, by a process similar to that used in the game Twenty Questions. The workspace (or context) contains a simple list of symbols, called "context list" (CL). "On-CL X" means that the symbol X is currently in the context. Figure 3-8 shows the rule base and the interpreter for our example production system.

PRODUCTIONS:

P1. IF ON-CL green THEN Put_On_CL produce
P2. IF On-CL packed in small container THEN Put-On-CL delicacy
P3. IF On-CL refrigerated OR On-CL produce THEN Put-On-CL perishable
P4. IF On-CL weighs 15 lbs AND On-CL inexpensive AND NOT On-CL perishable THEN Put-On-CL staple
P5. IF On-CL perishable AND On-CL weighs 15 lbs THEN Put-On-CL turkey
P6. IF On-CL weighs 15 lbs AND On-CL produce THEN Put-On-CL watermelon

INTERPRETER:

1. Find all productions whose condition parts are TRUE and make them applicable.
2. If more than one production is applicable, then deactivate any production whose action adds a duplicate symbol to the CL.
3. Execute the action of the lowest numbered (or only) applicable production. If no productions are applicable, then quit.
4. Reset the applicability of all productions and return to S1.

Figure 3-8. Productions and Interpreter
[Barr & Feigenbaum, 81]

The condition part of each of the productions corresponds to a question one might ask in the Twenty Questions game. Is the item green? Does it come in small container? and so on. The action parts of the productions represent addition to our knowledge about the unknown item.

Suppose the original knowledge about the mystery food item is that it is green and weighs 15 lbs. The context list before the beginning of the first cycle is

CL = (green, weighs 15 lbs.)

The cycle starts with step1 of the interpreter algorithm, finding all the applicable productions by testing their condition parts. Since only P1 is applicable, step2 is not necessary, and step3 causes the action part of P1 to be executed. This adds the symbol "produce" to the context list, representing a new fact about the unknown food item:

CL = (produce, green, weighs 15 lbs.)

Step4 ends the first cycle and brings us back to step1 - finding all the applicable productions.

In the second cycle, productions P1, P3, and P5 are all applicable. So in step2, we must check if any of these three adds a duplicate symbol to the context list. P1 adds "produce", which is a duplication, so it is eliminated. Then in step3 we select P3 to be executed because it has a lower number than P6. Now the CL looks like

CL = (perishable, produce, green, weighs 15 lbs.)

In the third cycle, P1, P3, and P5 are applicable. Checking, in step3, for redundant entries, we eliminate P1 and P3 from consideration. In step3, P5 is executed and watermelon is added to the context. The resulting CL is

CL = (watermelon, perishable, produce, green, weighs 15 lbs.)

In the last cycle, finding no non-redundant productions to execute, the interpreter finally quits. The system's answer is watermelon, because it is the first symbol on the context list.

3.2.5.6 Example 2

The next example is a PS that assists the service representative and mechanics in an automobile repair agency (see Section 2.2, "A Hypothetical KBS"). The example is based on [Barnett & Bernstein, 77].

A customer comes to the agency and reports the problems (and symptoms) to the Service Representative (SR). The SR enters the data into the system. The system diagnoses the problem(s) and suggests appropriate tests and repairs. The mechanic corrects the problem.

The system, as was mentioned in Section 2.2, contains

(1) Knowledge base of production rules that describe cause-and-effect relationships among the performance characteristics and measurable attributes of an automobile.

(2) A data base of past problems, repairs, and service performed on the vehicle.

Figure 3-9 shows a sample of production rules for the system.

RHS of each production rule has a condition, followed by decimal number which represents the certainty or probability of the condition (see Section 3.1.6, "Credibility Factors"). Thus, rule R1 says that, if the tension of the fan belt is low, then there are two possible consequences:

(1) That about one-half of the time the output of the alternator will be low.

(2) About one-fifth of the time the engine will overheat.

The other rules, R2 - R9, are interpreted in a similar manner.

R1 IF fan belt tension is low
THEN alternator output will be low [.5] and engine will overheat [.2]

R2 IF alternator output is low
THEN battery charge will be low [.7]

R3 IF battery is low
THEN car will be difficult to start [.5]

R4 IF automatic choke malfunctions OR automatic choke needs adjustment
THEN car will be difficult to start [.8]

R5 IF battery is out of warranty
THEN battery charge may be low [.9]

R6 IF coolant is lost OR coolant system pressure cannot be maintained
THEN engine will overheat [.7]

R7 IF there is a high resistance short AND fuse is not blown
THEN battery charge will be low [.8]

R8 IF battery fluid is low
THEN battery will boil off fluid [.3]

R9 IF battery fluid is low
THEN battery charge will be low [.4]

Figure 3-9. PRODUCTION RULES FOR AUTOMOTIVE SYSTEM KS

Figure 3-10 shows a fact file, a collection of "hard data". The information included for each measure or observation is the agent from whom to gather data and the relative difficulty (or cost) of gathering the data. There are four possible agents for gathering:

(1) The customer (Cust).

(2) The data base.

(3) Inspection by the service representative (SrvR).

(4) Measurement by the mechanic (Mech).

The difficulty information will be combined with the CFs in the production rules to formulate the most cost-effective and timely plan for the needed diagnostics and repairs.

Now assume that a customer comes to the agency with a vague complaint that his car is hard to start. The service representative enters this information, including appropriate customer and vehicle identification. The system then grows a structure similar to that shown in Figure 3-11. The boxes are labeled with observable or measurable symptoms and are connected by arrows labeled with the names of the production rule they represent. To the far right of each of the unknown value (e.g., the box labels, such as battery fluid level), the associated agent and relative difficulty are listed.

OBSERVATIONS	AGENT	DIFFICULTY
Alternate Output Level	Mech	4
Battery Charge Level	Mech	3
Battery Fluid Level	SrvR	2
Choke Adjustment	Mech	5
Choke Function	Mech	5
Coolant Level	SrvR	2
Coolant System Pressure	Mech	5
Difficulty to Start	Cust	1
Engine Temperature	Cust	1
Fan Belt Tension	Mech	3
Fuse Condition	SrvR	2
Short in Electric System	Mech	8
Voltage Regulator Level	Mech	4
Warranties	Data Base	0

Figure 3-10. DATA GATHERING PROCEDURE FACT FILE

At this point, the system would check the data base for information about the battery's warranty. If nothing decisive was found, then the customer would be asked whether the car was running hot, and the service manager would continue to make on-the-spot observations. Diagnostic procedures will then be placed on an ordered schedule for the mechanic. The ordering would be based upon :

(1) Cost effectiveness - a function of test difficulties, estimated probability of being necessary, and ability to eliminate other tests.

(2) Availability of resources - specialty mechanics and test equipment.

The structure shown in Figure 3-11 was grown by an algorithm called "back-chaining". A condition - in this case, "difficult to start" - is taken as a given, and the goal of the system is to find the cause(s).

The back-chaining algorithm is

(1) Find all rules that have the initial or derived conditions as their consequence (in our example, Rule R3 and R4).

(2) Call LHS (antecedents) of these rules - "derived conditions".

(3) Repeat steps (1) and (2), and terminate when no more can be done.

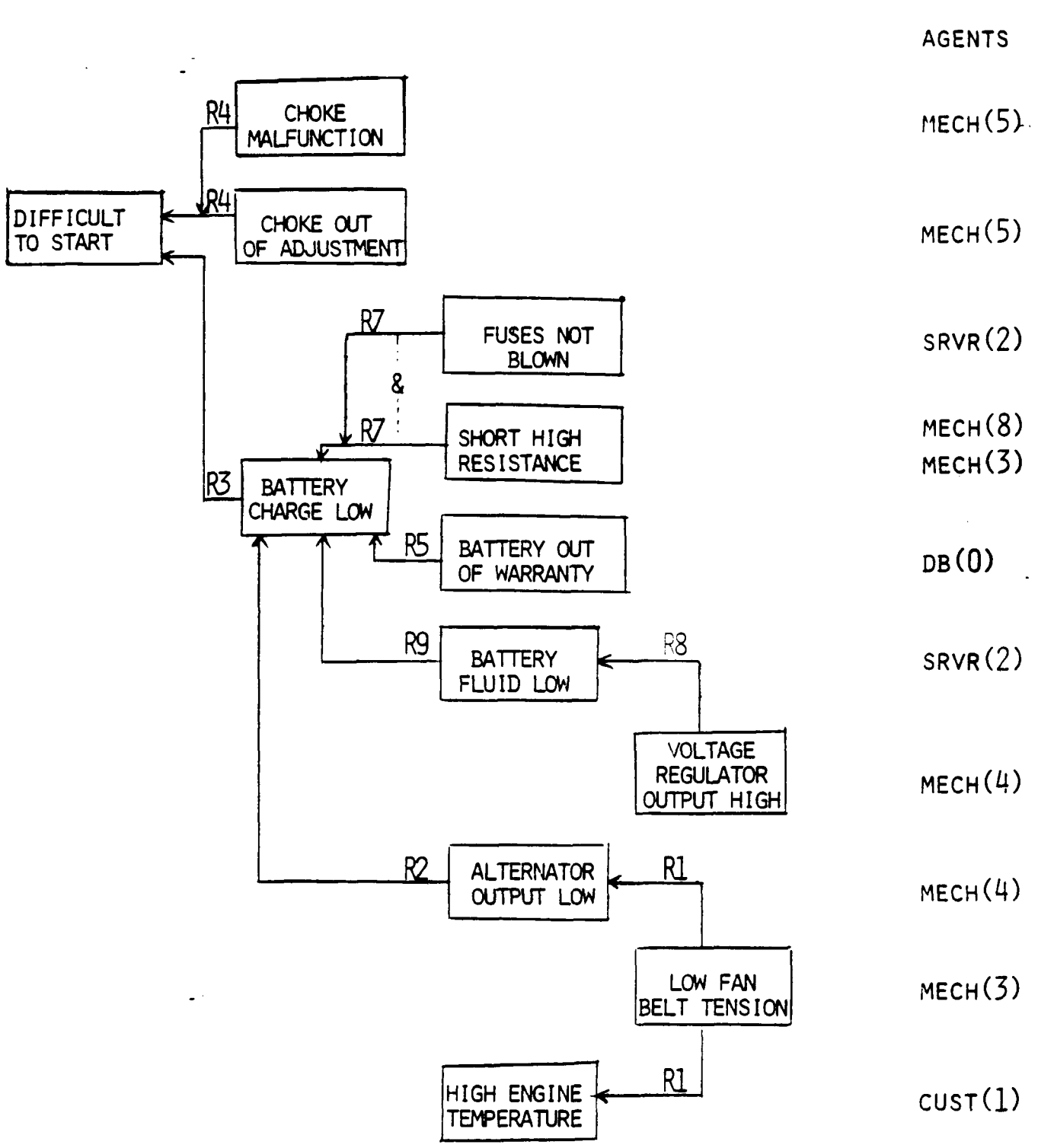


FIGURE 3.11 EXAMPLE FLOW IN AUTO DIAGNOSTIC SYSTEM

Figure 3-12 shows the kind of structure grown for each kind of rule format. In each example in the figure, C1 is the initial or a derived condition.

Rule E1 is the simplest; a1 is added to the set of derived conditions. Rule E2 states that if a1 is the case, then both C1 and C2 ought to follow. Thus, a1 is a derived condition, and C2 may or may not be considered a derived condition depending upon the particular strategy used by the system.

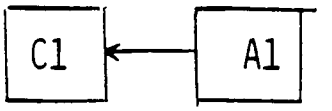
Rule E3 can be written as two rules: "IF a1 THEN c1" and "IF a1 THEN c2". Therefore, a1 is added to the set of derived conditions, and c2 part is ignored.

Rule E4 states that both a1 and a2 must occur to support the conclusion, c1. Therefore, both are derived conditions. If either a1 or a2 is found to not hold, then the search for support for the other can be discontinued.

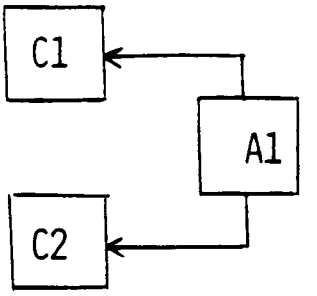
Rule E5 is equivalent to the separate rules "IF a1 THEN c1" and "IF a2 THEN c2". Thus, both a1 and a2 are added to the set of derived conditions.

The example and the discussion is somewhat simplistic because there might be some problems which we did not consider. For example, suppose that rule R8 (in Figure 3-9) had been written more accurately as the two rules:

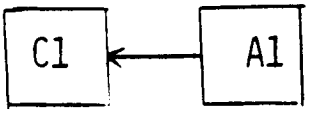
E1 IF A1 THEN C1



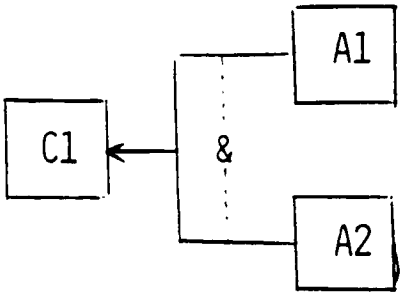
E2 IF A1 THEN C1 AND C2



E3 IF A1 THEN C1 OR C2



E4 IF A1 AND A2 THEN C1



E5 IF A1 OR A2 THEN C1

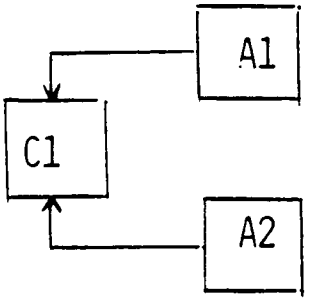


FIGURE 3.12 BACK CHAINING

R8(1) IF voltage regulator output is high
THEN the battery will overcharge.

R8(2) IF battery is overcharged
THEN battery will boil off fluid.

With these new rules, a fragment of the structure shown in Figure 3-11 would be replaced by that shown in Figure 3-13. Now the interesting conclusion is that a high battery charge implies a low battery charge. This is an apparent contradiction, since both conditions cannot hold at the same time. This kind of situation can often arise in unpredicted ways if the system contains many rules. The charge of the battery will oscillate between high and low as the battery fluid is replaced and boils off, respectively.

So, in a sense, there is a missing rule of the form that adding fluid to a battery whose charge and fluid levels are low will probably allow the battery to return to normal conditions. However, to handle this kind of situation in general, it is necessary that the control mechanism or inference engine have some knowledge about how to proceed when faced with apparent conflicts and contradictions. One advantage of PS is that ad hoc knowledge may be relatively easily incorporated in the system to handle this.

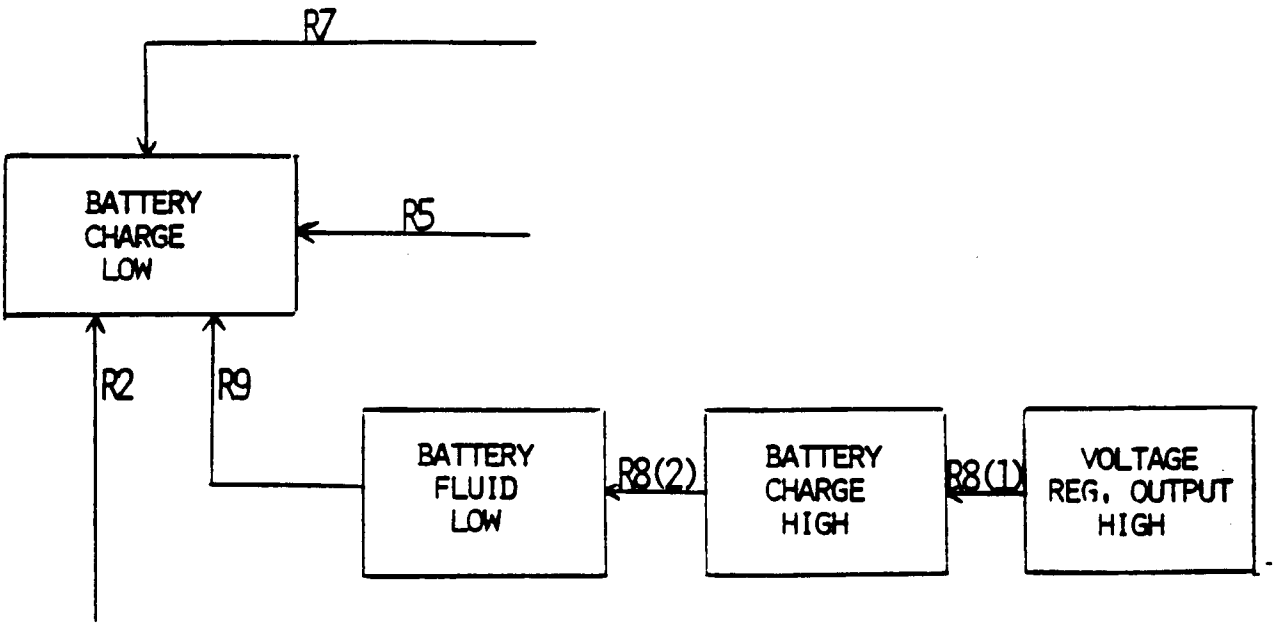


FIGURE 3-13. FRAGMENT OF GRAPH STRUCTURE

3.2.5.7 Characteristics of Production Systems

This section discussed some of the key features and characteristics of the production systems. The discussion is based on [Davis & King, 77] and [Barnett & Bernstein, 77].

Figure 3-14 is a summary of characteristics and relationships. Each box represents some feature, capability, or parameter of interest. An arrow labeled with "+" means that the source characteristic enhances the destination characteristic; the opposite is true for arrows labeled with a "-".

(a) Rules as Primitive Actions

In a production system, individual productions in the rule base can be added, deleted, or changed independently. Each production (or production rule) is a knowledge chunk.

(b) Indirect Limited Interaction Channel

One of the most fundamental characteristics of a production system is that production rules must interact indirectly through a single channel (or workspace). Rules are constrained to see and modify only the workspace. They cannot "call" each other. Thus, to produce a production system with a specified behavior, one must use an indirect approach in which each piece of code (i.e., each rule) leaves behind the proper traces (a unique message) to trigger the next relevant piece.

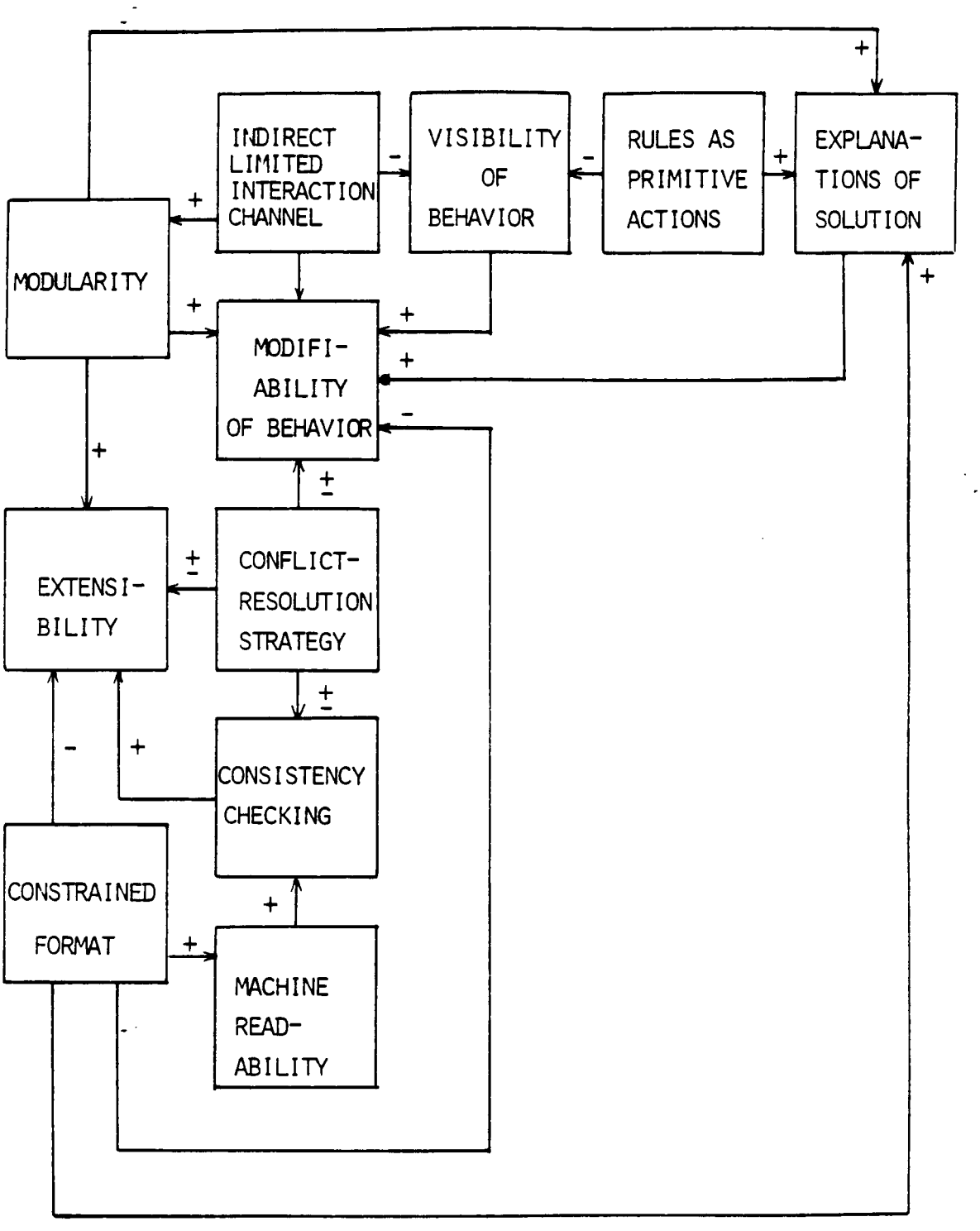


FIGURE 3-14. CHARACTERISTICS OF PRODUCTION SYSTEMS BASED ON [BARNETT & BERNSTEIN, '77]

The uniform access to the channel, along with openness of production systems (i.e., any rule could possibly be the next to be selected), implies that those traces (or messages) must be constructed in the light of a potential response from any rule in the system. This becomes more difficult to do as the number of rules increases and is a method that quickly destroys the major benefits of using PSs, such as independence of the knowledge chunks.

(c) Constrained Format

The syntax of production rules is traditionally quite restrictive. This means that:

(1) The LHS should be a simple predicate built out of Boolean combination of computationally primitive operations.

(2) The RHS should perform conceptually simple operations on the workspace.

Even though some systems allow programmer-supplied predicates and procedures to be invoked by the rule's LHS and RHS, some restrictions are obeyed [Davis & King, 77]:

(1) As a predicate, the LHS of the rule should return only some indication of the success or failure of the match.

(2) The operation of LHS must only "observe" the workspace, and not change it in the operation of testing it.

(3) The operation of RHS is precluded from using more complex control structures like iteration or recursion within the the expression itself (such operations can be constructed from multiple rules, however).

These constraints on form make the dissection and understanding of productions by other parts of the program a more straightforward task, strongly enhancing the possibility of having the program itself read, and/or modify its own procedures. Expressability suffers, however, since the limited syntax may not be sufficiently powerful to make expressing each piece of knowledge an easy task. This in turn, both restricts extensibility (adding something is difficult if it is hard to express it), and makes modification of the system's behavior more difficult. For example, it might not be particularly attractive to implement a desired iteration if it requires several rules rather than a line or two of code.

(d) Machine Readability

Constrained format enhances machine readability and allows the system to examine its own rules. As one example, it becomes possible to implement automatic consistency checking. Another capability deals with the MYCIN's approach to examining its

rules. This is used in several ways and produces both a more efficient control structure and precise explanations of system behavior [Davis, 76].

(e) Modularity

Since direct interaction among rules is constrained, it is possible to modify rules, delete rules, and add new rules as necessary because other rules are not directly dependent upon the rules that are changed or added.

For systems using the goal-directed (e.g., MYCIN) approach, rule order is usually unimportant. Insertion of a new rule is thus simple, and can often be totally automated. This is a distinct advantage where the rule set is large, and the problems of system complexity are significant.

(f) Extensibility

Extensibility is a corrollary of modularity. The ability to augment the system to perform in an expanded domain is obviously enhanced by the modularity and low interaction among the original rule set. On the otherhand, as was mentioned above under "Constrained Format", extensibility may be hampered because of format constraints if the expanded domain necessiates the use of a more robust set of primitives.

rules. This is used in several ways and produces both a more efficient control structure and precise explanations of system behavior [Davis, 76].

(e) Modularity

Since direct interaction among rules is constrained, it is possible to modify rules, delete rules, and add new rules as necessary because other rules are not directly dependent upon the rules that are changed or added.

For systems using the goal-directed (e.g., MYCIN) approach, rule order is usually unimportant. Insertion of a new rule is thus simple, and can often be totally automated. This is a distinct advantage where the rule set is large, and the problems of system complexity are significant.

(f) Extensibility

Extensibility is a corrollary of modularity. The ability to augment the system to perform in an expanded domain is obviously enhanced by the modularity and low interaction among the original rule set. On the otherhand, as was mentioned above under "Constrained Format", extensibility may be hampered because of format constraints if the expanded domain necessiates the use of a more robust set of primitives.

(g) Visibility of Behavior

Visibility of behavior is the ease with which the overall behavior of a production system can be understood, either by observing the system, or by reviewing its rule base. Even for conceptually simple tasks, the stepwise behavior of a production system is often rather opaque. The main factor responsible for this is the reevaluation of the workspace at every cycle. Because of these, any attempt to "read" a production system requires keeping in mind the entire contents of the workspace, and scanning the entire rule set at every cycle. Another factor is the limit on rule-to-rule communication which inhibits the system from focusing attention.

One method of increasing goal directed behavior in a production system is the use of high level, strategic and tactical rules to guide the conflict resolution strategy [Davis, 76]. An interesting discussion relating to this section can be found in [Englemore & Nii, 77].

(h) Modifiability of Behavior

This is similar to extensibility. However, the issue is the ability to modify the rules so that the system focuses attention better or more quickly. This is aided by modularity of the rule set and hindered by the problems that arise when explicit control and sequencing are desired in a production system.

(i) Explanation of Solution

A production system can (and usually does) explain and validate its solutions to problems by displaying the rules it used to derive the solutions. Because the rules are of a situation/conclusion form and are of reasonable chunk size, all necessary contextual information can be included in the rule itself. Modularity of the rules also contributes to the acceptability of the explanation because each rule is reasonably well self-contained.

(j) Conflict Resolution Strategy

Conflict resolution strategy has an effect on the ability to extend the system and/or modify its behavior. A RHS scan with backward chaining seems to be the easiest to follow since it mimics part of human reasoning behavior, while a LHS scan with a complex conflict resolution strategy makes the system generally more difficult to understand. As a result, predicting and controlling the effects of changes in or additions to, the rule base are directly influenced in either direction by the choice of rule selection method.

(k) Consistency Checking

If the rule set generates inconsistent results, the control mechanism may fail. Machine processing and simplicity of format help implement automatic consistency checking.

The best example of a KBS which uses production systems for representing knowledge is MYCIN.

3.2.6 Semantic Networks

3.2.6.1 Introduction

Semantic networks are used in many areas: psychological modeling of human memory, programming languages, natural language understanding, data base management systems, etc. And as such there is no simple set of unifying principles to apply across all semantic network systems.

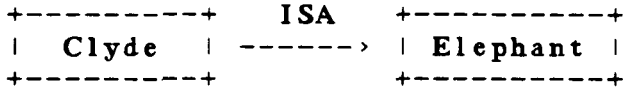
This section presents some general characteristics of semantic networks and illustrates some basic concepts with an example.

3.2.6.2 Definition

A semantic network (or net) consists of nodes and links (or arcs) and is a method of representing declarative knowledge. The nodes represent entities or objects, concepts or situations in the domain and the arcs represent the relations between them. Semantic networks, because of their inherent generality and naturalness, can be used to represent highly interrelated information that cannot be properly represented by, for instance, standard data (base) management techniques.

3.2.6.3 Example 1

Suppose we want to represent a simple sentence like "Clyde is an elephant" in a semantic network (example is taken from [Barr & Feigenbaum, 81]). We can represent this by creating two nodes Clyde and Elephant and connecting them with a link, as shown below.



This can also be written as

ISA(Clyde, Elephant)

It means that (Clyde, Elephant) is a member of the relation ISA. ISA (also known as "IS", "SUPERC", "SUPERSET") is conventionally taken to be the relation, more-specific-example-of. Thus the above example is the representation of the fact that Clyde is a specific example of Elephant.

Brachman [Brachman, 83] catalogs many other interpretations of ISA and differences between systems that, on the surface, appear very similar.

3.2.6.4 Example 2

Figure 3-15 shows another semantic network. In Figure 3-15(a) instances of various relations using the relation names TEMP, LOC, COLOR, SIZE, ISA, and BETWEEN are shown. The meaning of the relations is as follows:

TEMP(a,b) means a is the temperature of b.

LOC(a,b) means a is located at b.

COLOR(a,b) means that a is the color of b.

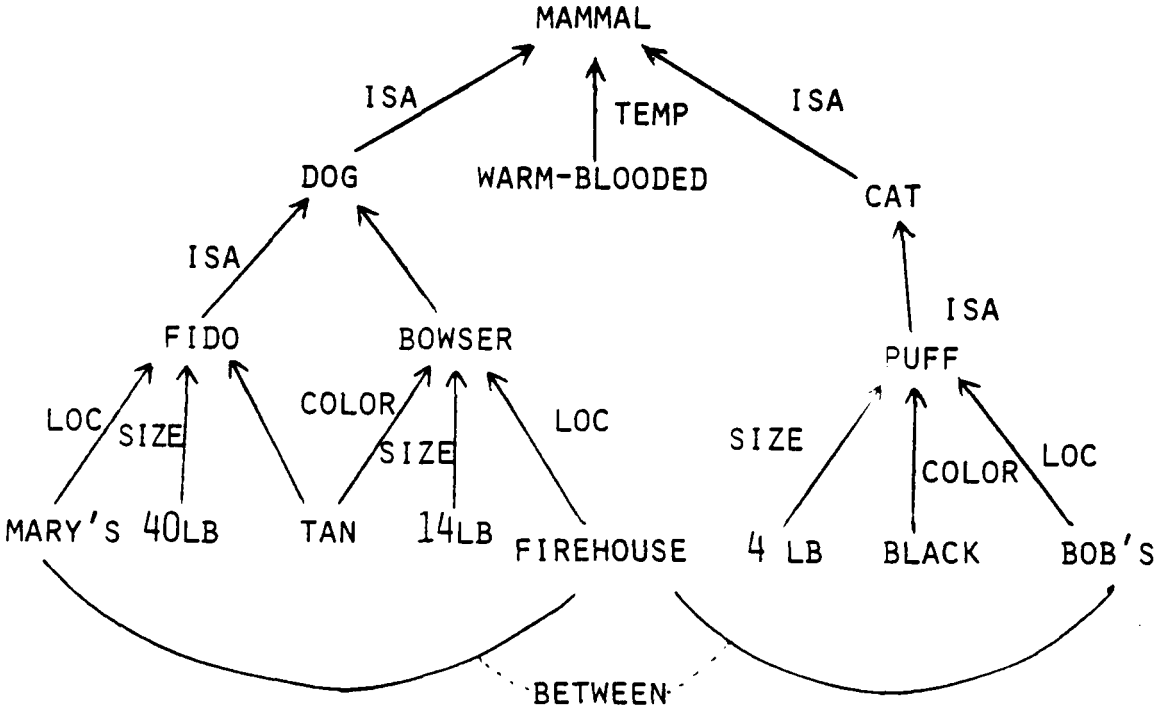
SIZE(a,b) means a is the size of b.

BETWEEN(b,a,c) means b is between a and c.

RELATIONS

TEMP(WARM-BLOODED MAMMAL)
ISA(DOG,MAMMAL) ISA(CAT,MAMMAL)
ISA(FIDO,DOG) ISA(BOWSER,DOG) ISA(PUFF,CAT)
LOC(MARY'S,FIDO) LOC(FIREHOUSE,BOWSER) LOC(BOB'S,PUFF)
COLOR(TAN,FIDO) COLOR(TAN,BOWSER) COLOR(BLACK,PUFF)
SIZE(40LB,FIDO) SIZE(14LB,BOWSER) SIZE(4LB,PUFF)
BETWEEN(MARY'S,FIREHOUSE,BOB'S)

SEMANTIC NETWORK



RULES OF INFERENCE

ISA(X,Y) ^ ISA(Y,Z) => ISA(X,Z)

SIZE(X,Y) ^ SIZE(U,V) ^ X<U => SMALLER(Y,V)

ISA(X,Y) ^ R(U,Y) => R(U,X)

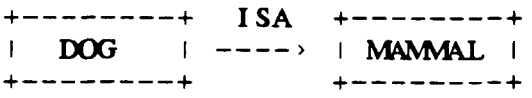
FIGURE 3.15 EXAMPLE SEMANTIC NETWORK

The knowledge in a semantic net is given meaning, as demonstrated here, by defining the relation names and other symbols used in the instances of relations, in terms of external entities.

Figure 3-15(b) shows a graph which represents the same knowledge that is in the set of instances shown in Figure 3-15(a). The object names are connected by arrows labeled with appropriate relation names. For example the instance

ISA(DOG, MAMMAL)

produces the graph fragment



Representation of graph fragments for other than binary relations is more difficult but still straightforward, for instance, BETWEEN in Figure 3-15(b).

The internal storage representation of semantic network is very similar to the graphical representation shown and is built using pointers and list structures. The explicit connections among the entities enhances the efficiency of programs that search through the semantic network [Barnett & Bernstein, 77].

Figure 3-15(c) shows some examples of inference rules for the semantic network. The format of the rules is well formed formulae from the predicate calculus (see Section 3.2.4). Inference rules can also be represented as production rules in a

production system. Production systems can be used to represent some procedural knowledge that can be used to test for complex enabling conditions. This may be difficult to express as WFFs. In Figure 3-15(c), variables, written as small letters, are assumed to be universally quantified.

The first rule says that (for all X, Y, and Z) if X is a Y and Y is a Z, then X is also a Z. An example of this is: PUFF is a CAT and CAT is a MAMMAL; therefore, PUFF is a MAMMAL. Thus first rule says that ISA is transitive.

The second inference rule says that if Y and V are two objects that "have" SIZE, and the size of Y is less than the size of V, then Y is SMALLER than V. For example,

SIZE(4,PUFF) & SIZE(14,BOWSER) & 4 < 14 => SMALLER(PUFF,BOWSER).

Thus second rule defines a new relation SMALLER, whose instances do not appear explicitly in the semantic network (Figure 3-15(b)).

The third inference rule says that, if X is a Y, and U is R-related to Y, then U is also R-related to X. For example,

ISA(FIDO, DOG) & ISA(DOG, MAMMAL) => ISA(FIDO, MAMMAL)

ISA(FIDO, MAMMAL) & TEMP(WARM_BLOODED, MAMMAL) =>

TEMP(WARM_BLOODED, FIDO)

Now let us consider the following example:

ISA(DOG, MAMMAL) & ISA(CAT, MAMMAL) => ISA(CAT, DOG).

This is a valid (by the application of inference rule 3) but erroneous inference. To avoid this kind of problem, it is necessary to have some non-syntactic (e.g., semantic) knowledge about the relations to which inference rules can be applied.

One solution is to embed the inference rules in the inference engine along with the necessary ad hoc knowledge to avoid problems.

Another solution is to have a rule, like the third one in the Figure 3-15(c), for each relation that is inheritable. However, both these solutions will cause problems, if the number of relations occurring in the semantic network is large or if the relation set can be modified or expanded.

A more general approach, originally proposed by Simmons and Slocum [Simmons & Slocum, 72], is to treat relation names and object names more uniformly. With this approach, relations can be arguments to relations, and hence have the same properties as other objects. For example, temperature is defined as an inheritable property by an instance like

INHERITABLE(TEMP)

The third inference rule in the Figure 3-15(c) can then be rewritten as

$$ISA(x, y) \;\&\; r(u, y) \;\&\; INHERITABLE(r) \Rightarrow r(u, x)$$

One advantage of this approach is that it provides a natural method of delineating legal values in a relation and, therefore,

it enhances error detection and consistency checking. Another advantage is improved flexibility and expandability. The major disadvantage of this approach is its loss in run-time efficiency.

Another choice and tradeoff in a semantic network is storage space and computation time. This arises from the decision about which relations and which instances in the relations should be stored explicitly and which should be computed via the inference rules. The number of instances of relations can grow in a highly non-linear way; for the example in Figure 3-15(b), the number of instances of the relation, SMALLER, grows as a quadratic function of the number of DOGs and CATs.

3.2.6.5 General Knowledge Versus Specific Knowledge

A technique often used with semantic networks is to make a distinction between general knowledge and specific knowledge and to store the two in a different manner. Referring to Figure 3-15(b) one can observe that specific knowledge lies at a low level in the tree. This means [Barnett & Bernstein, 77]:

(1) There are few, if any, chains below it.

(2) Properties have simple values.

(3) Most objects in the same general classification have all and only a known set of properties.

(4) There are large number of objects in a general class.

The specific knowledge in our example can be displayed as

ENTITY	ISA	SIZE	COLOR	LOC
FIDO	DOG	40 lb	Tan	Mary's
BOWSER	DOG	14 lb	Tan	Firehouse
PUFF	CAT	4 lb	Black	Bob's

The advantage of dividing knowledge into general and specific is that:

(1) The specific knowledge can be gathered into a tabular form, as shown above, by simple mechanical means.

(2) The specific knowledge (which is usually most of the semantic net) can be kept in relatively inexpensive secondary storage and even accessed through an efficient, existing data management system.

(3) The general knowledge can be kept in primary memory and, because most processing by the inference rules occurs on other than "bottom" of the network, efficiency can be maintained.

3.2.6.6 Advantages and Disadvantages

Advantages:

(1) Semantic nets can be used to represent definitional and relational knowledge that is too complex for ordinary data management techniques.

(2) Semantic networks allows inclusion of ad hoc information.

Disadvantages:

(1) The main disadvantage of using semantic networks to represent knowledge in KBSs is that the chunk size is fairly small. This causes two problems:

(a) Instances of relations do not lend themselves to being used in explanations of chains of reasoning developed by the inference rules - chains can be quite lengthy and tedious.

(b) Processing a semantic net can assume large amounts of computer time.

(2) Another disadvantage is that many kinds of knowledge (e.g., procedural knowledge, relative knowledge, etc.) cannot be expressed as instances of relations in a natural manner.

An example of KBS which uses semantic nets to represent knowledge is PROSPECTOR [Duda, et al, 78].

3.2.6.7 Status of Semantic Network Representation

Semantic nets are very popular knowledge representation methods in AI applications. Object-and-link structures capture something essential about symbols and pointers in symbolic computation [Barr & Feigenbaum, 81].

But processing non trivial nets can consume large amounts of computer time. Besides these problems, there are more subtle problems involving semantics of the network structures [Barr & Feigenbaum, 81]:

- What does a node (object) really mean?
- Is there a unique way to represent an idea?
- How is the passage of time to be represented?
- How does one represent things that are not facts about the world but rather ideas or beliefs?
- What are the rules about inheritance of properties in networks?

Current research on network representation schemes attempts to deal with these and similar concerns.

3.2.7 Frames

3.2.7.1 Introduction

There is abundant psychological evidence that people use a large, well coordinated body of knowledge from previous

experiences to interpret new situations in their everyday cognitive activity [Barr & Feigenbaum, 81]. How can we represent this type of knowledge in a computer system (program)? Many of the techniques of AI applications (programs) are not powerful enough to approach human performance in relation to vision, language, and common sense.

Minsky [Minsky, 75] first proposed a theory of "frames" as a mechanism for representing knowledge in the computer. His paper has evoked a great deal of discussion and interest in exploring further about frames and its theory. Some common motivating issues for this interest in frames are:

(1) Accommodation of both declarative and procedural knowledge in the same representational formalism.

(2) Accommodation of mundane, ad hoc, and idiosyncratic knowledge along with that which is more uniform and repetitive in nature.

(3) Accommodation of partial and somewhat contradictory or inconsistent knowledge.

(4) Ability to plausibly reason from a knowledge base with features like the above.

Two major issues not yet dealt with within the emerging theory of frames are explanation of system behavior and naturalness of the knowledge-acquisition interface.

3.2.7.2 Frame Characteristics

Some of the desirable features of frames are given below (Kuipers [Kuipers, 77] calls them a "wish list"). No single frame based system has all the desirable properties and it may be many years before the technical problems implied by such a frame theory (like the development of large-scale organization of knowledge, and the ability of these structures to provide direction for active cognitive processing [Barr & Feigenbaum, 81]) can be precisely stated and solved. The following discussion is based on [Kuipers, 77].

(a) Description

A frame provides an elaborate structure for creating and maintaining a description of an object in a domain. And as such a frame can be viewed as a single knowledge chunk. The description of an object includes a number of features of that object and the relations which hold among those features.

A frame has named slots corresponding to those definitional characteristics (i.e., features, relations, etc.). A primitive element in a frame may be expanded to another frame and/or procedural knowledge may be attached to an element when it's internal description becomes of interest.

(b) Instantiation

This is the process by which the frame creates a description

from observation of an object in its domain. Features whose real properties have not been observed are represented by default (or assumed) values. These default values can be static or computed in terms of the values in other slots.

(c) Prediction or Expectation

A frame's predicted (or expected) description can be used to guide the collection of observations for instantiation. It also produces the defaults which substitute for unobserved features.

(d) Justification

Different features of the frame description have different amounts of confidence. Some are clear observations, others are choices among a few alternatives, and others are default assignments.

(e) Variation

A frame represents a certain (limited) domain, and hence a range of variation for objects which belong to that domain is limited and specified. When a feature (or set of features) of a frame is outside the permissible range of variation in a frame, it may cast doubt on the applicability of this frame and may indicate to the correction mechanism that another mechanism is called for.

(f) Correction

In most common cases of recognition, the identity of the object being described is not initially known. So selecting the proper frame to instantiate is part of the problem. The current "best guess" frame attempts to create a correspondence between what it expects to see and the observations actually available.

Anomalies may indicate that the current frame is not correct, and that a different point of view is called for. The frame can analyze the anomoly to select a more appropriate replacement. The procedures that test and deal with unusual conditions are called monitors.

(g) Perturbation

For small changes in the observer or the observed, perturbation procedures correct the description without complete recomputation.

(h) Transformation

In case of more significant changes, transformation procedures propose frames suitable for the new situation. Those experiences - the experiences that lead to those significant changes - are saved (by complaint procedures) and incorporated into newer versions of the "faulty" frames when structural revisions become possible.

3.2.7.3 Example 1: Frame Representation

It is not possible to give a simple example that has all the above properties of a frame. The following example (Figure 3-16, based on [Barnett & Bernstein, 77]) is provided to illustrate some of the concepts involved in frame based systems.

The top of the Figure 3-16(a) provides a description about a dog. Explanation for each line is provided below (line numbers are not part of frame definition; they are provided for explanation purposes only).

Line 1: The first line states that a dog is a mammal.

Line 2: Line 2 means that there is a slot named "kind" (of dog), that may be filled with a type of "breed". "Breed" is itself a frame.

Line 3: The color of the dog is limited to one or a combination of the colors selected by the SUBSET.OF operator.

Line 4: The FROM operator is used to pick out values from other frames and default values are indicated by underlining. Thus the combined effect of the phrase FROM Color OF Kind is to make the default value for the color of a dog the default for his breed.

Line 5: Line 5 means that there is a slot for the number of legs and the range is 0 to 4 with a default of four.

Line 6: Line 6 represents a slot for weight, which is a positive integer with a default that is determined by

the typical size of members of the same breed.

Line 7: The state of the dog is either "adult", the default, or "puppy", if age is known to be less than one year.

Line 8: The age of dog is restricted to be a positive number and its default value can be calculated procedurally by "now birthday".

Line 9: The birth date of the dog is represented as a date in this slot.

Line 10: The name of the dog is represented as a string in this slot.

Line 11: The end of description of dog frame.

Figure 3-16(b) shows a frame for "boxer".

```
1   dog       FRAME   ISA   mammal
2             kind    breed
3             color   SUBSET.OF {tan brown black white rust}

4             FROM color OF kind

5             leggedness        0...4
6             weight            >0,  FROM size OF kind
7             state             adult  OR  puppy if age < 1
8             age               >0,  now birthday
9             birthday          date
10            name              string

11            END               dog
```

(a)

```
1   boxer     FRAME   ISA   breed OF dog
2             color             ONE.OF {tan brown brindle}
3             size              40...60
4             tail              bobbed OR long
5             ears              bobbed OR floppy
6             temperment        playful
7             COMPLAINTS        IF weight > 100 THEN ASSUME
                                              (great dane)
8             END               boxer
```

(b)

Figure 3-16. EXAMPLE FRAME DEFINITIONS
[Barnett & Bernstein, 77]

Line 1: Line 1 declares that boxer is a breed and it is a dog.

Line 2: The color of a boxer is restricted to one of the colors tan, brown, and brindle, with a default of tan. It is legal for this to conflict with the dog frame (Figure 3-16(a)); i.e., brindle is not mentioned in that frame. If this breed did not have a color restriction, then this slot would be omitted; this would have the effect of not giving a default assignment for color in the dog frame (in Figure 3-16(a)).

Line 3: This slot says that the size of a boxer is between 40 and 60 pounds. No default is specified. Thus when the dog frame is applied to boxer, this default range will be used for weight (rather than an exact value).

Line 4: This slot says that tail can be "bobbed" or "long" with "bobbed" being the default.

Line 5: The ears can be either "bobbed" or "floppy" with "bobbed" being the default.

Line 6: Line 6 says that temperament is always playful.

Line 7: This is an example of a complaint and ad hoc knowledge used to make a recommendation, namely, if you see a giant boxer (> 100lbs.), then assume that it might be a Great Dane instead.

Line 8: End of description of boxer frame.

3.2.7.4 Example 2: A Recognition Scenario

Procedures can be attached to slots to recognize (or reason) a task. In some frame based systems, attached procedures are the principal mechanisms for directing the reasoning process, being activated to fill in slots or being triggered when a slot is filled [Bobrow, 79].

Filling Slots

After a particular frame has been selected to represent the current context or situation, the primary process in a frame based system is often just filling in the details called for by its slots.

Figure 3-17 shows an example use of frame in a recognition task. The top of the figure (Figure 3-17(a)) shows some feature values that have been detected for an object, here identified as 654.

A general matching procedure would attempt to instantiate all frames in the system until a reasonable fit was found; in our example, "boxer" is a reasonable match. Then the slots in the boxer frame will be filled with the observed data. If data is not available, default values will be used. If there is no contradiction, procedural attachments will be used to decide the values for the slots.

```
LOW-LEVEL  INFORMATION

      OBJECT  654

          color       = tan
          ears        = bobbed
          leggedness  = 4
          size        = 40 - 45
          temperment  = mean

TRIAL  IDENTIFICATION

      [OBJECT  654 ISA dog

          kind        boxer   WITH [color       tan
                                    size        40 - 45
                                    tail        ASSUMED bobbed
                                    ears        bobbed
                                    temperment  EXCEPTIONAL
                                                mean]
          color       tan
          leggedness  4
          weight      40 - 45
          state       ASSUMED adult]
```

Figure 3-17. INEXACT MATCH BY A FRAME SYSTEM
[Barnett & Bernstein, 77]

Default values are relatively inexpensive method of filling slots; they do not require powerful reasoning process. These methods account for a large part of the power of frames - any new frames interpreting the situation can make use of values determined by prior experience, without having to recompute them. When the needed information must be derived, attached procedures can take advantage of the current context, namely, slot-specific heuristics. In other words, general problem-solving methods can be augmented by domain-specific knowledge about how to accomplish specific, slot-sized goals.

In our example, after filling the color and size slots, as information for the tail slot is not available, a bobbed tail will be assumed (assuming there was a frame for tails).

Similarly, when it tries to fill the temperment slot, it notes the observed feature, "mean", which is a contradiction to the expected value "playful". Thus, it activates the complaint mechanism which notes that this particular boxer (object 654) is mean and it is exceptional.

If the weight of the boxer was too large, the complaint mechanism could (tentatively) change the identification of the instantiation of the boxer into the one for a Great Dane. There are two advantages to this:

(1) Rather than returning to a very general pattern-matching activity, a candidate that is highly likely to be right is selected next.

(2) The slot values for this frame can be transferred to the new frame with little additional work.

If the match is good enough, then the frame can become more informative. In our example, the transformation is from boxer to boxer dog, where more information is observed, e.g., leggedness. Also, the dog is assumed to be adult.

The above steps (prediction, correction, and gathering of more information) continue until all of the low level information is consumed. The belief is that the style of recognition will be more goal directed and hence more accurate and efficient than general techniques that depend upon regularity and uniformity of structure.

3.3. Inference Engine

3.3.1 Primary Functions of Inference Engine

The IE provides central control of the KBS and thus affects both the performance and power of the system. The functions of the IE are: knowledge use and control, knowledge acquisition, and explanation. To do these, the IE must:

(1) Control and coordinate system activities and resources.

(2) Plausibly reason about domain specific problems by having access to and using the contents of the

knowledge base, the contents of workspace, and knowledge and procedures embedded in the IE.

(3) Link the KB with the inference module(s).

As was mentioned in Section 2.5.1, in a KBS, the ability to solve a problem depends on:

(1) How many paths there are to a solution.

(2) The ability of the IE to reduce the number to a minimum.

(3) The knowledge in the KB.

(4) What information is available within the problem statement.

Therefore, although the IE is in command and acts as the driving element, the path to a solution and the criteria for which to accept a solution or abort a particular path are highly dependent on the content of the KB and the problem data.

In the next section, some terminology (definitions) to describe inference engines is presented. This terminology is based on [Nilsson, 81].

3.3.2 Definitions

Sound IE: A IE is sound if it produces only correct or "I don't know" solutions, i.e., it does not produce incorrect solutions.

Complete IE: A IE is complete if it can always produce a

solution to a posed problem when a solution exists.

Admissible IE: A IE is admissible if it always finds a minimal-cost solution when a solution exists. The cost is taken to mean the cost of using the solution, not necessarily the cost of finding it.

3.3.3. Inference Engine Control Strategies

In this section, some control strategies used by IEs are presented.

The input to an IE is usually a set of initial conditions (or states) and goals. The IE uses the KB and one of the control strategies to obtain the goal(s), operating within the constraints imposed by the initial conditions.

Some of the control methods are discussed below:

3.3.3.1 Forward Chaining

This method involves applying the KB to the given conditions to infer new conditions; continue in this manner until the goal is satisfied. This strategy is also called data-driven, event-driven, and bottom-up (see the example in Section 3.2.5). The rules applied to a state to produce new states are called F-rules.

3.3.3.2 Backward Chaining

This method involves applying the KB to the goal (or goal description) to produce new subgoals; continue this manner until constraints or primitive conditions (known to be solvable) are reached. Backward chaining is also known as goal-driven, expectation driven, and top-down. The rules applied to produce goals to produce subgoals are called B-rules.

3.3.3.3 Chain Both Ways

This method involves forward chaining from the initial conditions and backward chaining from the goal until a common middle term is produced, i.e., F-rules are applied to initial state and B-rules are applied to goal states. The control mechanism must, at every state, decide whether to apply an applicable F-rule or an applicable B-rule.

3.3.3.4 Middle Term Chaining

This method involves using the KB, guessing a middle term and solving separately the problem of getting from the initial conditions to the middle term and from the middle term to the original goal. Continue in this manner until a solution in terms of primitives is generated. This method is also called problem reduction.

Figure 3-18 shows an example of first three techniques. The problem is to transfer 4 to 20. The KB contains three rules:

(1) Any integer, X, can be replaced by 2X (X -› 2X).

(2) Any even integer, 2X can be replaced by X (2X -› X).

(3) Any integer, X can be replaced by 3X + 1 (X -› 3X + 1).

Figure 3-18 shows the use of forward chaining. Start with 4 and apply the operators until 20 is produced.

Figure 3-18 shows the use of back chaining. Start with the goal, 20, and use the inverse of the above rules and continue until 4 is produced.

Figure 3-18 shows the use of the chain both ways technique. First, one step of back chaining produces the nodes labeled 10 and 40. Then one step of forward chaining produces the nodes labeled 8, 2, and 13. Finally, one more step of back chaining is done to produce the nodes labeled 5, 3, 13, and 80. Since 13 is in both the forward and backward grown "wave fronts", the process can terminate; otherwise, the steps of forward and backward chaining would continue until either a solution was found or the system gave up because of violation of some constraints (like computation time, for instance).

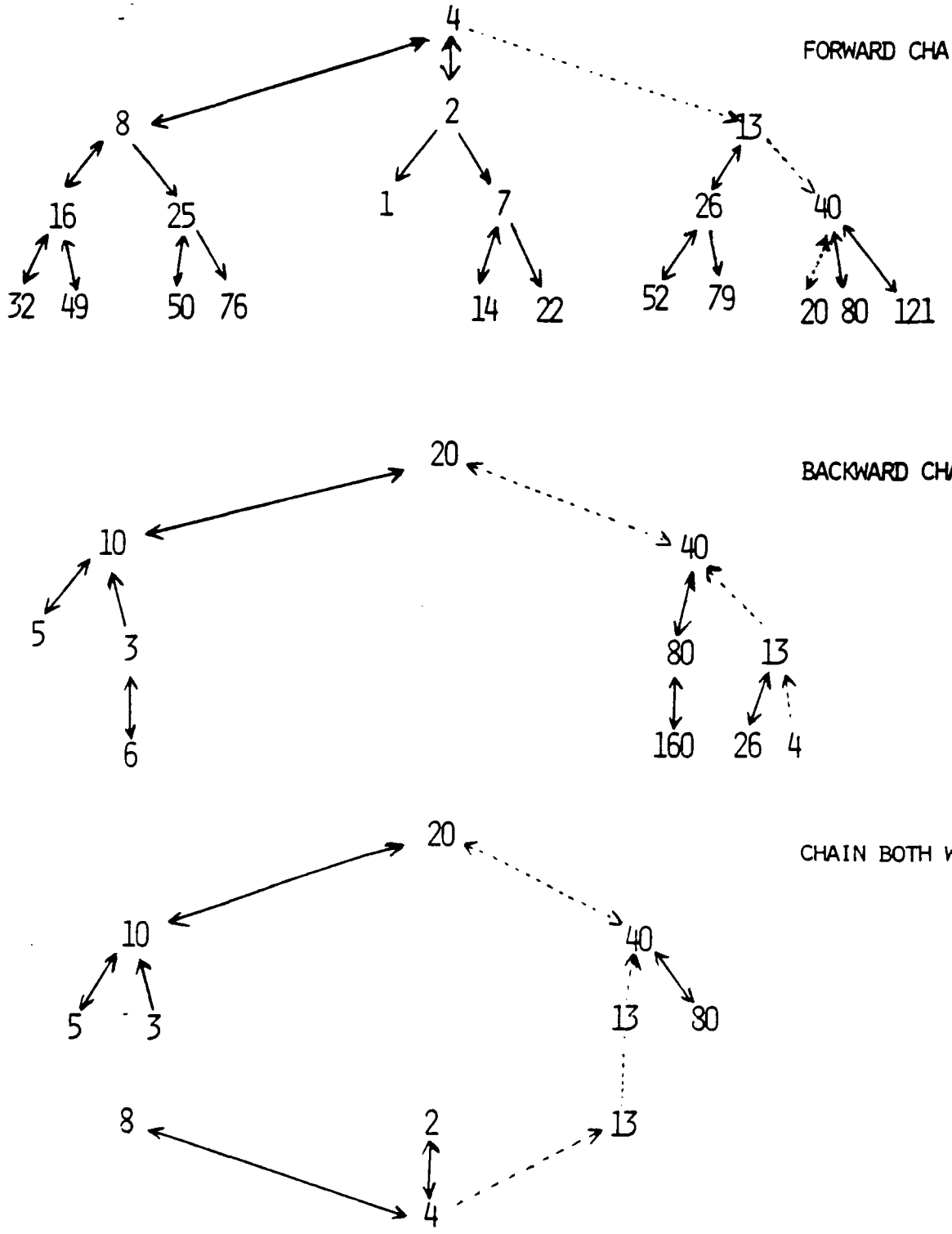


FIGURE 3.18 CHAINING EXAMPLES

Figure 3-19 shows an example of problem reduction approach. The problem is to show that AD equals CD. To show this, the problem can be reduced to the following subproblems [Nilsson, 71]:

(1) In order to show that two line segments are equal, show that they are corresponding elements of congruent triangles.

(2) In order to show that two triangles are congruent, show the equality of a side and two triangles in corresponding positions or of an angle and two sides.

(3) In order to show that two angles are equal, show that they are both right angles.

Of course, these problems could be further divided into primitive form. The actual proof of this problem can be found in [Nilsson, 71].

An example system, Gelernter's Geometry Theorem-Proving Machine [Newell & Simon, 72], uses this technique to solve a given problem.

GIVEN: ABD = CBD
AD ⊥ BA
CD ⊥ BC

PROVE: AD = CD

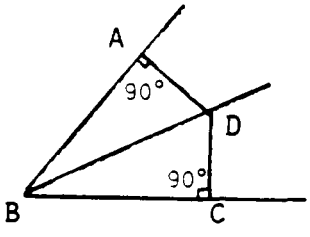


Figure 3-19. Diagram for Problem Reduction [Nilsson, 71]

3.3.3.5 Directionality of Control Strategy

Another way of classifying IE control strategies is by its directionality. This type of classification is typically used in speech understanding systems where the input (waveform) is linearly ordered. The two major types are: fixed directionality and variable directionality.

(a) Fixed Directionality

This type of control strategy is typically described as left-to-right or right-to-left. In the fixed directionality type of control strategies, the input is processed in a predetermined direction until either:

(1) All data have been consumed and the problem is successfully solved or

(2) A block is reached and no further progress can be made.

In the latter case the system reacts in a predetermined fashion, typically backing up to a point before the block occurred at which point an alternative option was available. At this point, an alternative path is assumed, and processing of the input is continued in the original direction. This technique is iterated until either the problem is solved or no more alternatives exist.

(b) Variable Directionality

The first problem in speech understanding systems is, given a sentence to understand where to start, starting with the first word in a sentence is not necessarily the most efficient strategy [Barr & Feigenbaum, 81]. The fixed-direction type of strategies work well with the precompiled network representation. The disadvantage of this strategy is that if the first word is not identified correctly, or is not identifiable, understanding the rest of the sentence is retarded. In such cases variable direction control strategy can be used.

A completely variable directionality in a system is often called island driving. The idea is to start processing the input at the point or points deemed to be least ambiguous or contain the most robust clues as to their identity. The points (also called anchor points or islands) are then grown, middle outward until they collide or a block occurs. If a block occurs, another set of points are determined in the unprocessed areas. Thus, by starting in areas containing the more certain information (more certain hypotheses), part of combinatorial explosion of fixed-directionality strategies will be avoided because back up will rarely occur across the islands, but only between them. A problem with the island driving strategy, however, is that there can be many islands and, hence, many hypotheses most of which may not be reliable and soon have to be abondoned.

This type of strategy is used in HEARSAY, and in the SRI

Speech Understanding System.

(c) Hybrid Strategy

Another strategy, explored in HWIM ("Hear What I Mean", developed by BBN) [Wolf & Woods, 80], is a hybrid between island driving and the left to right strategy. The problem of not being able to understand the first word in the sentence is overcome by trying to understand any of the first three or four words. Then the expansion of this word is in one direction at a time: first back to the beginning of the sentence, and then to the end. This dramatically reduces the number of extension hypotheses that must be considered at one time [Barr & Feigenbaum, 81].

Another way of differentiating IE strategies is via breadth-first vs. depth-first.

3.3.3.6 Breadth-First Control Strategy

In a breadth-first system, all possible methods of continuing are attempted in parallel. This is shown in Figure 3-18, where each (horizontal) level of the graph was generated by a single cycle of the system. The breadth-first strategy is described by the following algorithm (from [Nilsson, 71]):

(1) Put the start node on a list called OPEN. If the start node is a goal node, a solution has been found.
(2) If OPEN is empty, exit with failure; otherwise continue.

(3) Remove the first node on OPEN and put it on a list called CLOSED; call this node n.

(4) Expand node n, generating all of its successors. If there are no successors, go immediately to (2).

(5) Put the successors at the end of OPEN and provide pointers from these successors back to n.

(6) If any of the successors are goal nodes, exit with the solution obtained by tracing back through the pointers; otherwise go to (2).

An Example: 8-Puzzle.

An 8-puzzle is a square tray containing eight square tiles of equal size numbered 1 to 8. The space for the ninth tile is vacant. A tile may be moved by sliding it vertically or horizontally in to the empty square. The problem is to transform one particular configuration say, that of Figure 3-20(a), into another given tile configuration say, that of Figure 3-20(b).

Figure 3-20. 8-Puzzle

Figure 3-21 (taken from [Nilsson, 81]) shows the breadth-first strategy applied to an 8-puzzle. The nodes are labeled by their corresponding state description and are numbered

in the order in which they were expanded. The dark branches show a solution of five moves.

3.3.3.7 Depth-First Control Strategy

In a depth first system, some path (node, state, etc.) is selected and a single continuation is attempted, i.e., the node is not fully expanded all at once. This path continues growing until either the path reaches a solution or some path-length constraint is violated. In the latter case, the path is backed up to the deepest node at which an alternative expansion exists. At that point, another path is generated. This process continues until either a solution is produced or the alternatives are exhausted (Figure 3-22).

The depth of a node is defined as follows [Nilsson, 71]:

(1) The depth of the root node is zero.

(2) The depth of any node descendent of the root is one plus the depth of its parent.

The following algorithm describes the depth-first control strategy (taken from [Nilsson, 71]):

(1) Put the start node on a list called OPEN. If it is a goal node, a solution has been found.

(2) If OPEN is empty, exit with failure; otherwise continue.

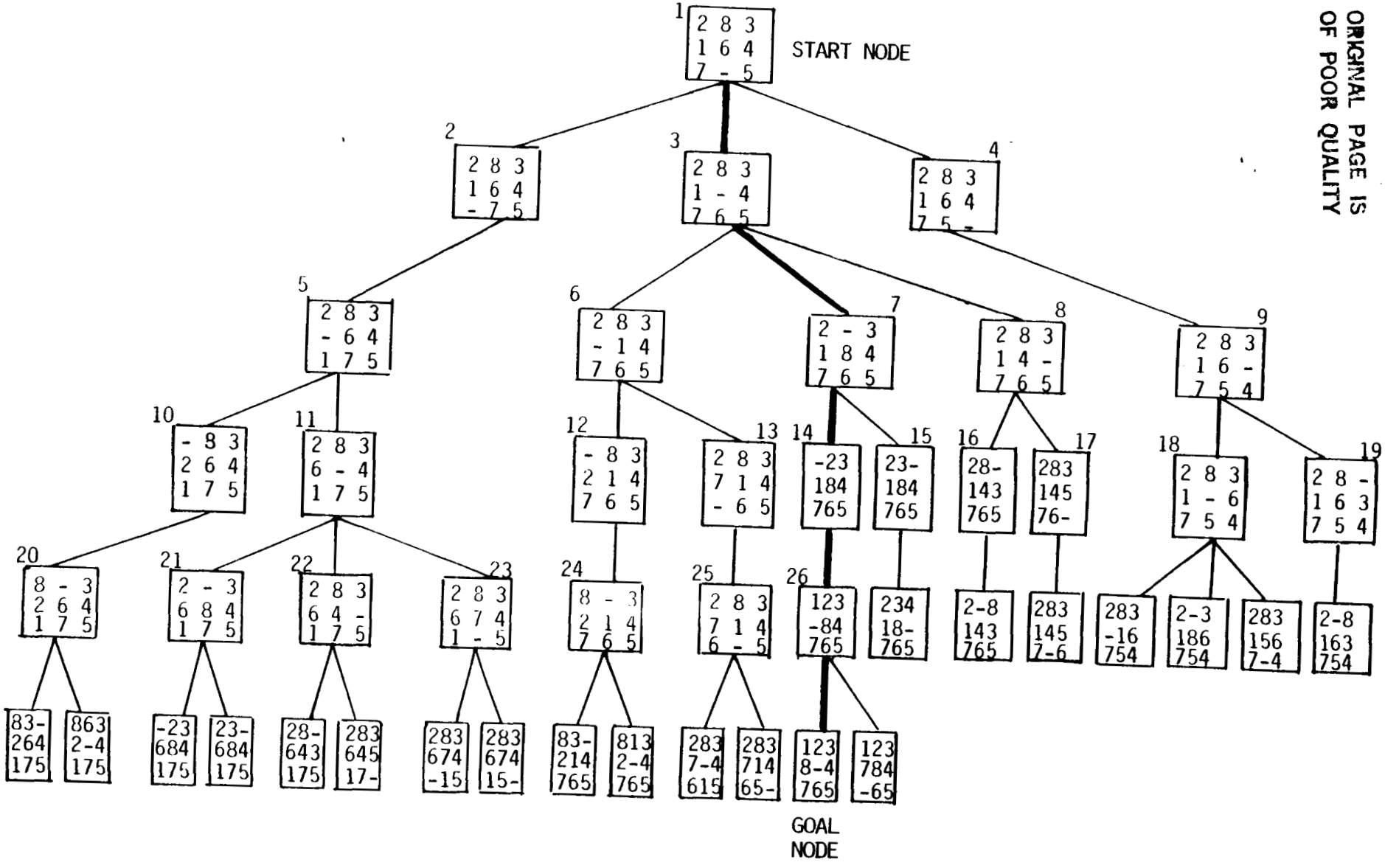


FIGURE 3-21. THE TREE PRODUCED BY A BREADTH-FIRST SEARCH
BASED ON [NILSSON, '71]

(3) Remove the first node from OPEN and put it on a list called CLOSED. Call this node n.

(4) If the depth of n equals the depth bound (maximum depth), go to (2); otherwise continue.

(5) Expand node n generating all successors of n. Put these (in arbitrary order) at the beginning of OPEN and provide pointers back to n.

(6) If any of the successors are goal nodes, exit with the solution obtained by tracing back through the pointers; otherwise go to (2).

An Example: 8-Puzzle

Figure 3-23 shows the tree generated by using depth-first strategy. The problem, like before, was to transform the configuration shown in Figure 3-20(a) into the configuration 3-20(b).

Figure 3-23 (taken from [Nilsson, 71]) shows depth-first control strategy applied to an 8-puzzle.

3.3.4 Methods of Implementing the Inference Engine

Most methods and techniques used to implement inference engines are restricted by the choice of a representation technique for the knowledge base (see Section 3.1.2, "Choices and Restrictions"). However, a few methods (e.g., search methods) are general enough to be used with a variety of knowledge base representations. In this section, some search techniques are discussed.

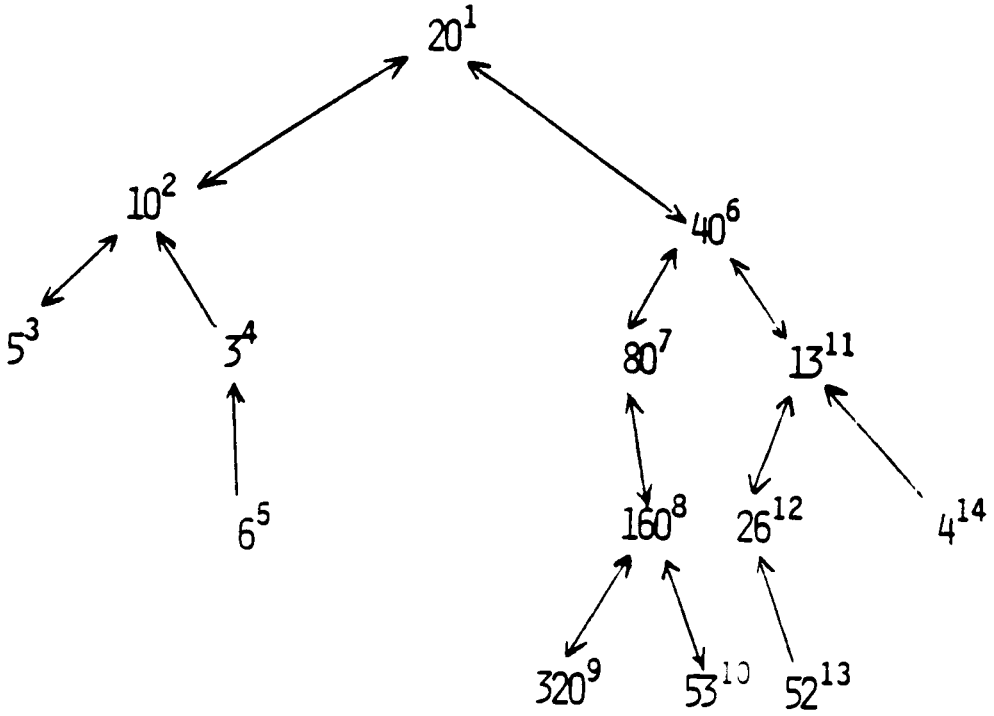


FIGURE 3-22. DEPTH-FIRST BACK CHAINING

3.3.4.1 Search Techniques

Search techniques used in KB and AI systems refer to a large body of core ideas that deal with deduction, inference, planning, common sense, and related processes. The real problem with search technology (or techniques) is:

(1) To find an algorithm with a specified set of characteristics, and

(2) To ensure that that algorithm is efficient and does not suffer from combinatorics when handling problems in the intended area of application. To accomplish this, it is necessary to incorporate domain-specific knowledge.

3.3.4.2 Search System Components

A search system consists of five major components:

(1) Select - pick the next activity to be performed from agenda of possible next activities.

(2) Expand - perform the selected activity, which often means enumeration of some or all of the predecessor activities.

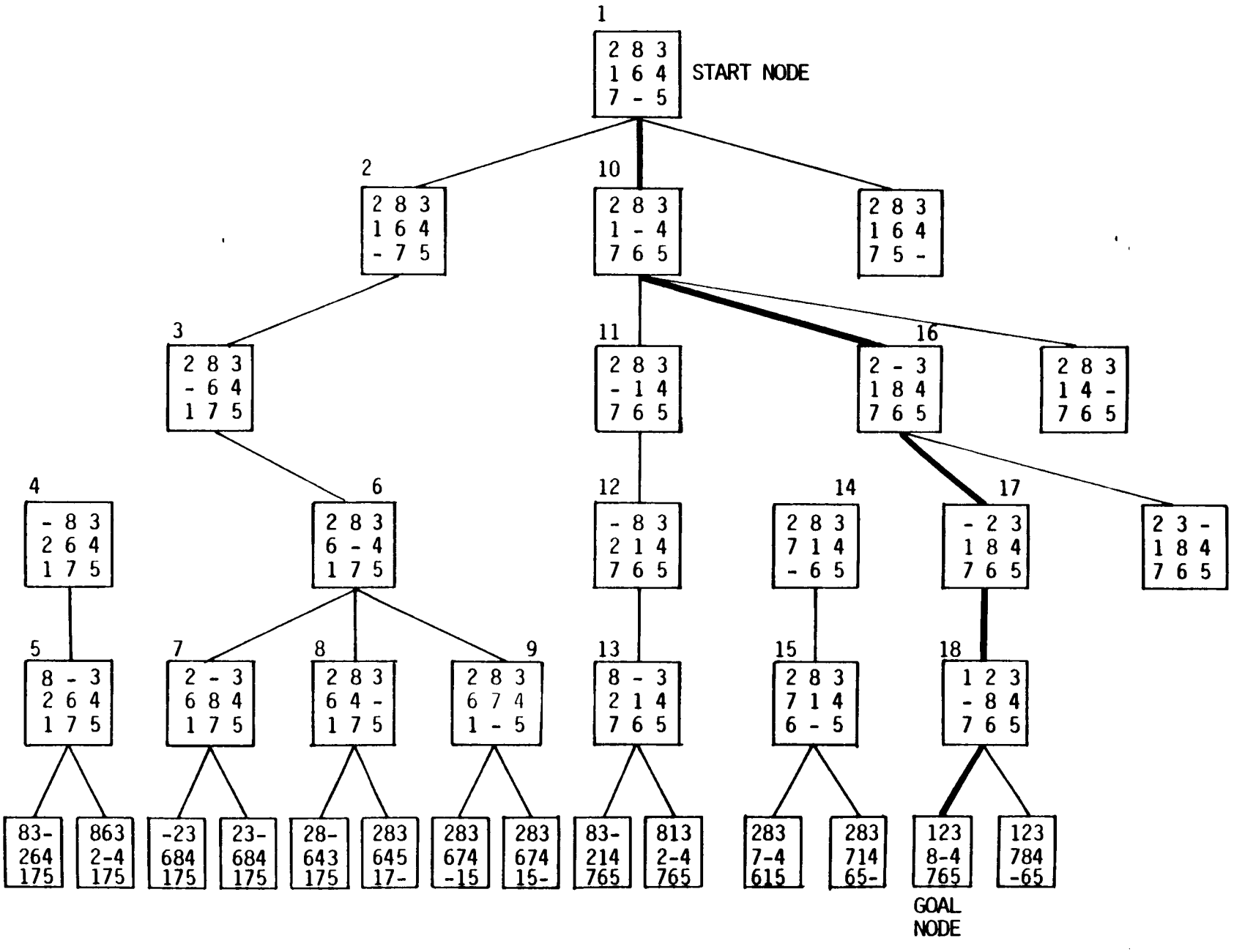


FIGURE 3-23. THE TREE PRODUCED BY A DEPTH-FIRST SEARCH
BASED ON [NILSSON, '71]

(3) Evaluate - compute merit scores for activities created by the expansion process.

(4) Prune - discard hopeless cases or those that appear to have little promise.

(5) Terminate - determine whether to continue processing and whether the problem has been sufficiently solved.

From the above five components, it is easy to realize the importance of knowledge base in providing accurate guidance for each component (by incorporating domain-specific knowledge). This could improve the system performance by orders of magnitude.

In many search methods, the selection, evaluation, and pruning (if any) are combined into a uniform numerical technique. The function used for this purpose is called an evaluation function.

3.3.4.3 Evaluation Function (f)

The purpose of an evaluation function is to provide a means for ranking those nodes (activities) that are candidates for expansion to determine which one is most likely to be on the best path to the goal [Nilsson, 71].

Suppose some function, f, could be used to order nodes for expansion, then f(n) denotes the value of this function. The evaluation function f is defined so that the more promising a node is, the smaller is the value of f. The node selected for expansion is one at which f is minimum.

Conventionally, the nodes are ordered in increasing order of their f values. An algorithm which selects a node (from a list of nodes called OPEN) having the smallest f value (for next expansion) is called an ordered-search algorithm i.e., an ordered-search algorithm selects a node for expansion at which f is minimum.

3.3.4.4 Ordered Search Algorithm

The ordered search algorithm (taken from [Barr & Feigenbaum, 81]) is given below.

(1) Put the start node s on a list, called OPEN, of unexpanded nodes. Calculate f(s) and associate its value with node s.

(2) If OPEN is empty, exit with failure; no solution exists.

(3) Select from OPEN a node i at which f is minimum. If several nodes qualify, choose a goal node if there is one, and otherwise choose among them arbitrarily.

(4) Remove node i from OPEN and place it on a list, called CLOSED, of expanded nodes.

(5) If i is a goal node, exit with success; a solution has been found.

(6) Expand node i, creating nodes for all its successors. For every successor node j of i:

a. Calculate f(j).

b. If j is neither in list OPEN nor in list CLOSED, then add it to OPEN, with its f value. Attach a pointer from j back to its predecessor i (in order to trace back a solution path once a goal node is found).

c. If j was already on either OPEN or CLOSED, compare the f value just calculated for j with the value previously associated with the node. If the new value is lower, then:

i. Substitute it for the old value.

ii. Point j back to i instead of to its previously found predessor.

iii. If node j was on the CLOSED list, move it back to OPEN.

(7) Go to (2).

The way in which the algorithm works is illustrated by considering the same 8-puzzle example.

An Example: 8-puzzle

Consider the simple evaluation function

$$f(n) = g(n) + w(n)$$

where g(n) is the length of the path in the search tree from the start node to node n, and w(n) counts the number of misplaced tiles in the state description with node n. Thus the start node

2	8	4
1	6	4
7	-	5

has an f value equal to 0 + 4 = 4.

The results of applying the ordered-search algorithm to the 8-puzzle and using this evaluation function are summarized in Figure 3-24. The value of each node is circled. The uncircled numbers show the order in which nodes are expanded. It is interesting to note that the same path is found here as was found by other search methods, although the use of evaluation function has resulted in substantially fewer nodes being expanded.

The search results are critically dependent on the choice of the evaluation function, f, which should discriminate sharply between promising and unpromising nodes. If the discrimination is inaccurate, however, the ordered search may miss an optimal

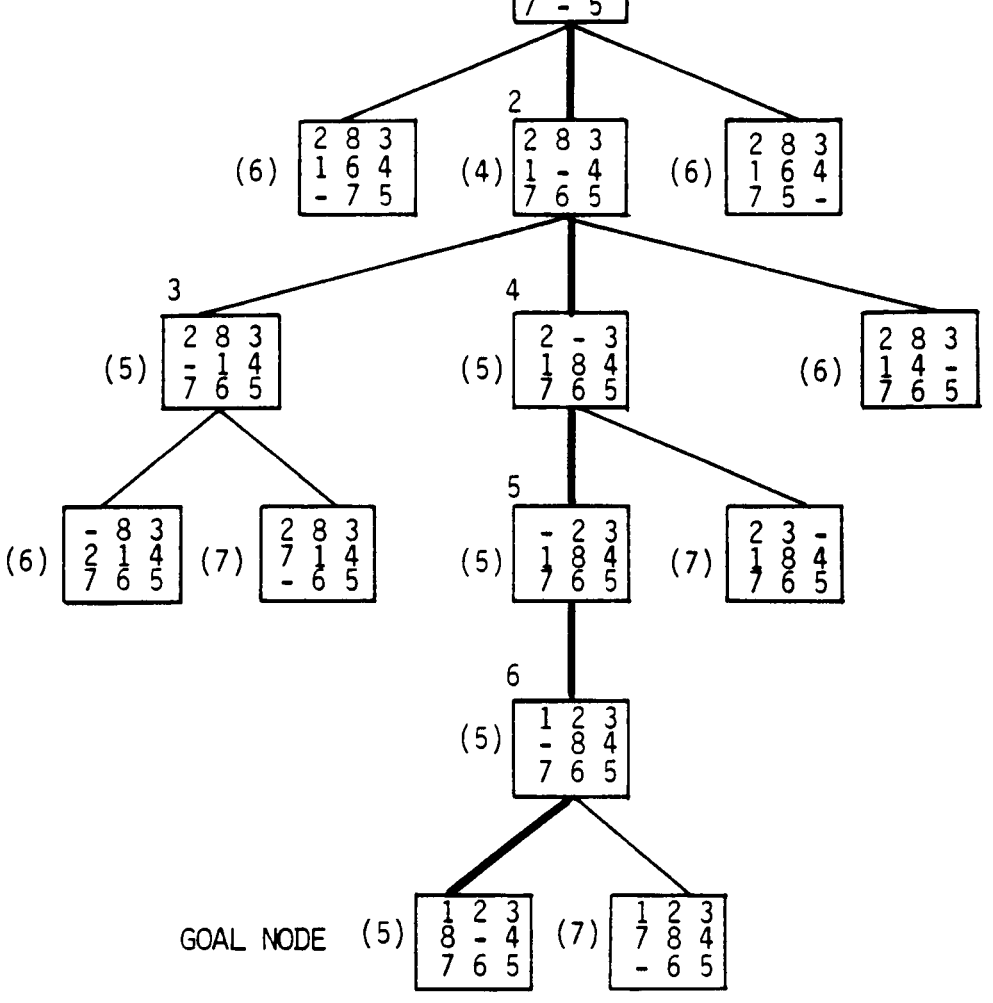


FIGURE 3-24. THE TREE PRODUCED BY AN ORDERED SEARCH
BASED ON [NILSSON, '71]

solution or all solutions. It can be proved [as in Nilsson, 71] that the ordered search algorithm is "sound" no matter how bad the evaluation function is.

In the next section one particular evaluation method which can produce an optimal (minimum cost) solution is described. It is called the A* algorithm.

3.3.4.5 A* = An Optimal Search Algorithm

The A* algorithm being described was proposed by Barr & Feigenbaum [Barr & Feigenbaum, 81]. Historically, the predecessors of A* include Dijkstra's [Dijkstra, 59] and Moore's [Moore, 59] algorithms. A class of algorithms similar to A*, is used in operations research under the name of branch-and-bound algorithms.

In A*, the evaluation function, f'(x) is the cost of a solution path constrained to go through node x; hence, its value is to be minimized. Further, f' is assumed to be additive in the cost of going from one node in a path to another. Thus, if n(1) .... n(m) (n(1) = start, n(m) = goal) is an optimal solution path, then

$$f'(n(i)) = \sum_{j=1}^{m-1} K(n(j), n(j+1)) \quad 1 <= i <= m$$

where K(x,y) is the cost of going from state x to state y in one step. For any node, n, f' can be expressed as

$$f'(n) = f'(start, n) + f'(n, goal)$$

where $f'(x,y)$ is the minimal cost of a path (of perhaps many steps) from x to y. Normally, the above is written as

$$f'(n) = g(n) + h(n)$$

where

$g(n) = f'(start, n)$ and

$h(n) = f'(n, goal)$.

We desire our evaluation function f to be an estimate of f'. Thus f can be approximated as

$$f(n) = g'(n) + h'(n).$$

Where g' is the estimation of g, and h' is the estimation of h.

The A* algorithm is given below [Hart, et al, 68]:

(1) Mark s "open" and calculate f(s).

(2) Select the open node n whose value of f is smallest. Resolve ties arbitrarily, but always in favor of any node n belonging to T (T is the set of goal nodes).

(3) If n belongs to T, mark n "closed" and terminate the algorithm.

(4) Otherwise, mark n closed and apply the successor operator S to n. Calculate f for each successor of n

and mark as "open" each successor not already marked closed. Remark as open any closed node n(i) which is the successor of n and for which f(n(i)) is smaller now than it was when n(i) was marked closed. Go to (2).

It can be shown that A* is admissible and optimal [as in Nilsson, 71]. To guarantee admissibility, a necessary condition is that

$$h'(n) <= h(n) \text{ for all } n.$$

A necessary condition for being optimal is that

$$h'(x) - h'(y) <= K(x,y).$$

This is called the consistency condition. Without this constraint, A* will be still be admissible but no longer optimal [Nilsson, 71].

3.3.4.6 AND/OR Graph

The discussion in the previous sections (including breadth-first and depth-first strategies) is related to what is generally known as state space search. The 8-puzzle is a simple example of state-space representation. This section discusses search methods in relation to problem-reduction.

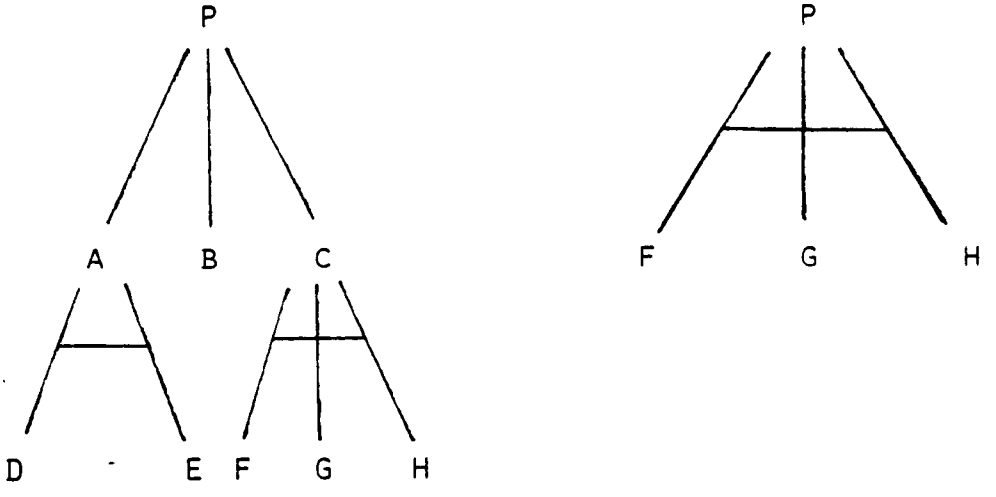
A generalized notation for problem reduction is called the AND/OR graph. According to Nilsson [Nilsson, 71], an AND/OR

graph is constructed according to the following rules:

(1) Each node represents either a single problem or a set of problems to be solved. The graph contains a start node corresponding to the original problem.

(2) A node representing a primitive problem, called a terminal node, has no descendants.

(3) For each possible application of an operator to problem P, transforming it to a set of subproblems, there is a directed arc from P to a node representing the resulting subproblem set. For example, Figure 3-25 illustrates the reduction of P to three different problem sets: A, B, and C.



a. b.

Figure 3-25. An AND/OR Tree

[Barr & Feigenbaum, 81]

(4) For each node representing a set of two or more subproblems, there are directed arcs from the node for the set to individual nodes for each subproblem. Since a set of subproblems can be solved only if its members can all be solved, the subproblem nodes are called AND nodes. To distinguish them from OR nodes, the arcs leading to AND-node successors of a common parent are joined by a horizontal line.

(5) A simplification of the graph produced by rules 3 and 4 may be made in the special case where only one application of an operator is possible for problem P and where this operator produces a set of more than one subproblem. As Figure 3-25 illustrates, the intermediate OR node representing the subproblem set may then be omitted.

A node or problem is said to be solved if one of the following conditions holds:

(1) The node is in the set of terminal nodes (primitive problems).

(2) The node has AND nodes as successors and all these successors are solved.

(3) The node has OR nodes as successors and any one of these successors is solved.

A node or problem is said to be unsolvable if:

(1) The node has no successors and is not in the set of terminal nodes. That is, it is a nonprimitive problem to which no operator can be applied.

(2) The node has AND nodes as successors and one or more of these successors are unsolvable.

(3) The node has OR nodes as successors and all of these successors are unsolvable.

The difference in searching an AND/OR graph and an ordinary state-space graph is the presence of AND. This causes many conceptual complications to the search problem.

Definition of an Optimal Solution

A solution of an AND/OR graph is a subgraph demonstrating that the start node is solved. The cost of a solution tree can be defined in either of two ways [Barr & Feigenbaum, 81]:

(1) The sum cost of a solution tree is the sum of all arc costs in the tree.

(2) The max cost of a solution tree is the sum of arc costs along the most expensive path from the root to a terminal node.

For example, if every arc in the solution tree has cost 1,

then the sum cost is the number of arcs in the tree; and the maximum cost is the depth of the deepest node.

Let C(n,m) be the cost of the arc from node n to a successor node m. Define a function h(n) by:

(1) If n is a terminal node (a primitive problem), then h(n) = 0.

(2) If n has OR successors, then h(n) is the minimum, over all its successors m, of c(n,m) + h(m).

(3) If n has AND successors and sum costs are used, then h(n) is the summation, over all successors m, of c(n,m) + h(m).

(4) If n has AND successors and max costs are used, then h(n) is the maximum, over all successors m, of c(n,m) + h(m).

(5) If n is a nonterminal node with no successors, then h(n) is infinite.

According to this definition, h(n) is finite if and only if the problem represented by node n is solvable. For each solvable node n, h(n) gives the cost of an optimal solution tree for the problem represented by node n. If s is the node, then h(s) is the cost of an optimal solution to the initial problem.

An example AND/OR tree is shown in Figure 3-26.

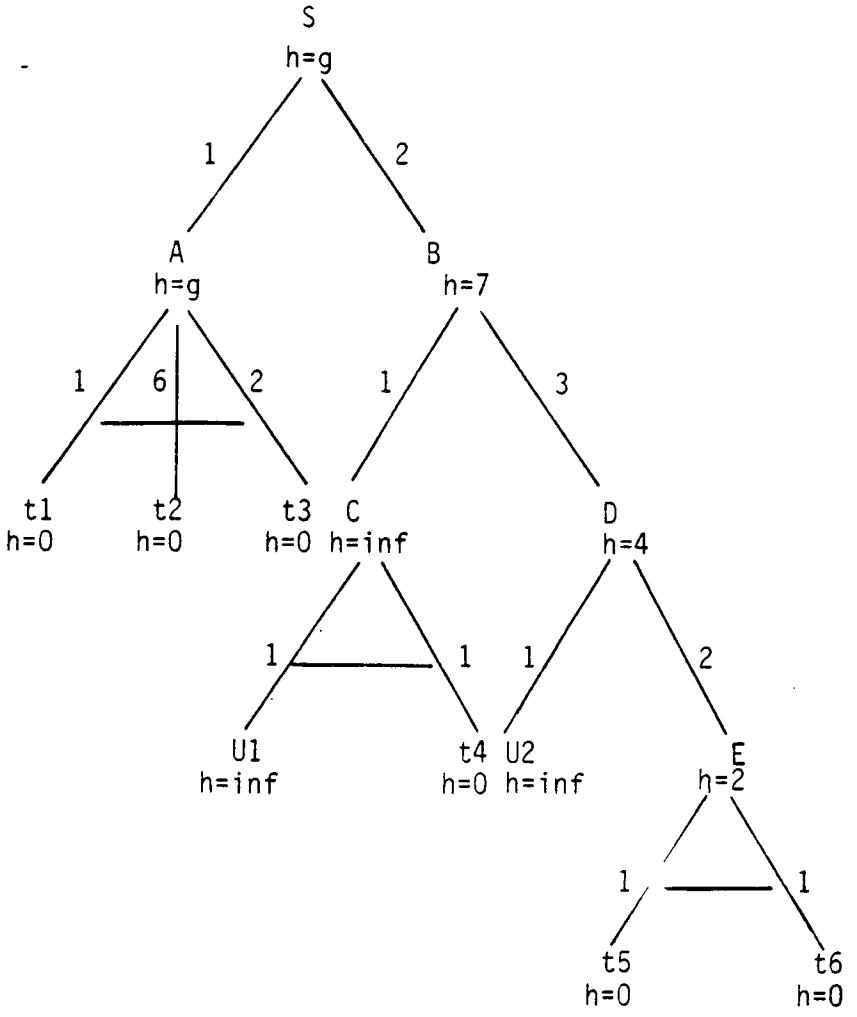


FIGURE 3-26. SUM COSTS
BASED ON [BARR & FEIGENBAUM, '81]

3.3.5 Measures of Performance

Section 3.3 is concluded by briefly discussing two measures of performance. The definitions and discussion are based on [Nilsson, 71] and [Barr & Feigenbaum, 81].

Performance measurement for KBSs is not easy as it is for many other types of computer systems, because run time and dynamic memory consumption of KBSs are often highly non-linear functions of some problem parameters. As a result it is very difficult to determine the heuristic power of a search technique in KBSs. However, certain measures of performance can be used in comparing various techniques. They are: penetrance and branching factor.

3.3.5.1 Penetrance

The penetrance, P, is defined as

$$P = L/T$$

where L is the length of the derived path from the initial state (or node) to the goal, and T is the total number of states (or nodes) generated while searching for a solution.

If the IE proceeds directly to a solution without generating any false paths or unused states, the penetrance achieves its maximum value 1. Blind search is characterized by small values of P. Since performance is usually nonlinear with L, the value of P generally decreases with increasing L, and the value of P(L)

is estimated to characterize performance.

3.3.5.2 Branching Factor

Branching factor is more nearly independent of the length of the optimal solution path. Its definition is based on the assumption of a tree with the same total number of nodes, T, as states produced by the system in solving a problem. The tree is further assumed to be one in which:

(1) Every expanded node has B descendants, and

(2) The tree has paths of length, L, the number of operators in the solution path of the original problem.

Therefore,

$$T = B + B^2 + \ldots + B^L = \sum_{i=0}^{L} B^i$$

This can be written as

$$T = (B^{L+1} - 1)/(B-1)$$

and solved for B, the branching factor, by iteration.

By definition, B can never be less than 1. A value of B near unity (i.e., small) corresponds to a search that is highly focused toward the goal with very little branching in other directions, while large values of B indicate that the system has wasted time expanding nodes not used in the final solution or has

included states that have not been further expanded.

3.3.5.3 Examples

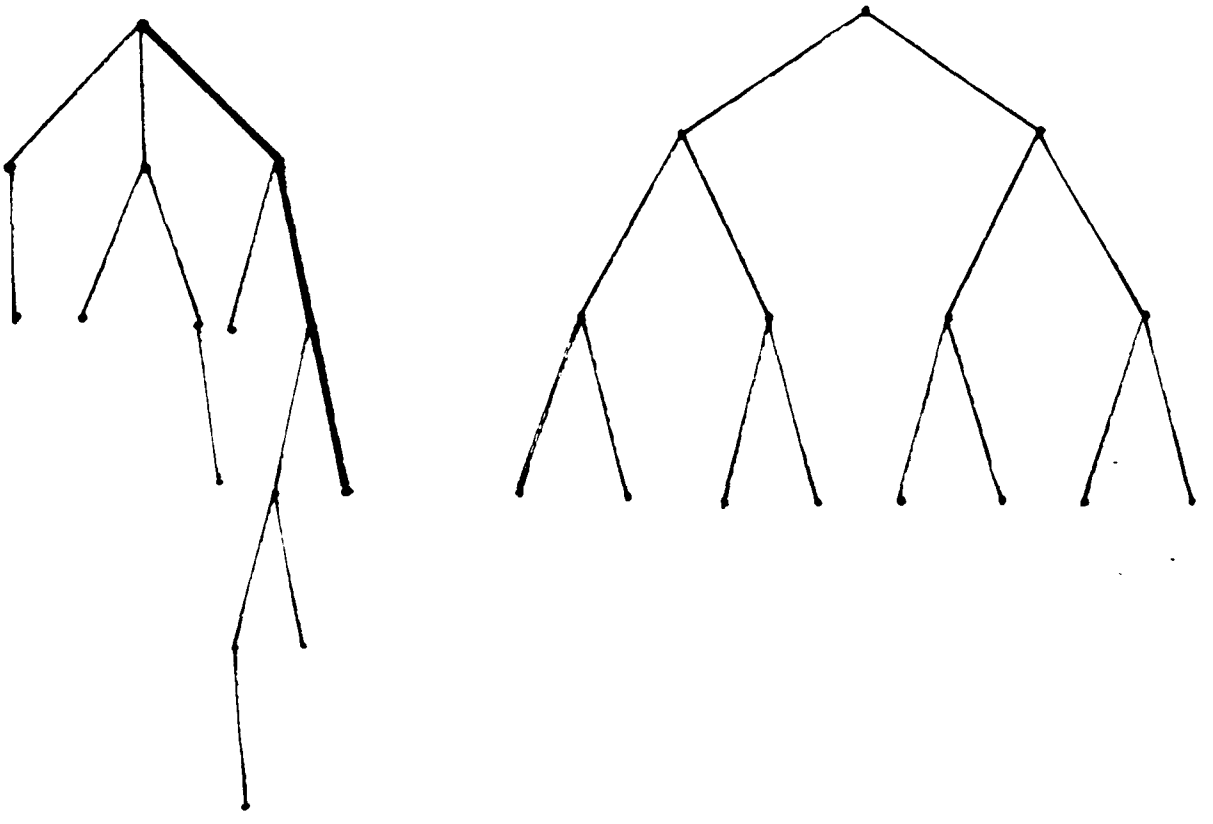
Figure 3-27 shows a graph with T = 15 nodes and a solution path (shown by the darkened line) of length L = 3. Therefore, the penetrance

$$P = L/T = 1/5.$$

To the right is shown a balanced tree with T = 15 and L = 3. As can be seen from Figure 3-27, B = 2 and one can verify that

$$T = (2^4 - 1)/(2-1) = 15.$$

This measure is useful in applications where computation time is a function of input length; for example, the number of words in a sentence or input to a natural language understanding system.



a. b.

T = 15

L = 3

P = 1/5

B = 2

FIGURE 3-27. EXAMPLE MOVE GRAPH AND BALANCED TREE

3.4 Workspace Representation (or Blackboard)

3.4.1 Introduction

One of the major component in a KBS is the workspace representation (or blackboard). A blackboard records intermediate hypotheses, decisions, and results that a KBS manipulates during a problem solving activity i.e., it is the encapsulation of the system's current state in a problem solving activity. It includes:

(1) Plan - the plan describes the overall or general attack the system will pursue against the current problem (including current plans, goals, problem states, contexts, etc).

(2) Agenda - the agenda is a list of activities that can be done next (which generally correspond to knowledge base rules that are relevant to some decisions taken previously).

(3) History - the history records what has been performed (and why) to bring the system to its current state, which is used to provide explanations.

(4) Solution Set - a solution set represents the candidate hypotheses and decisions the system has generated thus far (along with the dependencies that relate decisions

to one another).

A simple example of a workspace representation in a programming language system (like LISP) is a push-down stack. The stack contains the bindings of global variables, temporary values, and return addresses. In this type of system, the program counter (which identifies the instruction to be executed next) acts as the agenda mechanism. These systems, however, do not have any explanation mechanism, which is essential to a KBS.

Every KBS uses some type of workspace for intermediate decision representation, but only a few explicitly employ a blackboard for the various types of functions described above.

The following subsections briefly discuss two techniques used to represent workspaces : HEARSAY Blackboard (also known as CMU Blackboard) and Move Graphs (or AND/OR graphs).

3.4.2 HEARSAY Blackboard

The designers of HEARSAY-II within the Carnegie-Mellon University Speech Understanding Systems, employed a novel and interesting way to represent a workspace called a "blackboard" [Erman, et al, 80]. The same technique has been used in KBSs built for various tasks such as:

X-ray crystallography [Feigenbaum, et al, 77].

Signal interpretation [Nii & Feigenbaum, 78].

Vision [Hanson & Riseman, 78].

Psychological modeling [Rumelhart, 76].

The blackboard is a data structure:

(1) On which the hypotheses and their support criteria can be stored, and

(2) Which acts as an intermediary among multiple knowledge sources and the system's inference engine.

Knowledge in HEARSAY-II is organized into various knowledge sources. The board is subdivided into 8 information levels corresponding to intermediate representation levels of the decoding process (phrases, words, syllables, etc.). The primary relationships between levels is compositional: word sequences are composed of words, words are composed of syllables, and so on. Each hypothesis resides on the blackboard at one of the levels and bears a defining label chosen from a set appropriate to that level. When KSs are activated, they create and modify these hypotheses on the blackboard, record evidential support between levels (usually adjacent), and assign credibility ratings.

Figure 3-28 shows levels and KSs in the HEARSAY-II system. Arrows, labeled with KS names, show input (circled ends) and output (pointed ends) levels.

Figure 3-29 shows a fragment of a blackboard (a very simplified version of one presented in [Erman, et al, 80]). As depicted, the support is ambiguous. For example, the word ARE at the lexical level could be supported by the existence of the phonemes AW, ER at the phonetic level. Or the word ARE could have been predicted from higher level considerations and then caused the phoneme predictions. The Figure 3-29 also shows another competing word OR. This could have resulted if the phonemes AW, ER were ambiguously recognized as either ARE or OR. Then, the "ARE ANY" would be in competition with "OR ANY".

Thus, the blackboard serves as an ideal structure for representing competing hypotheses. HEARSAY-II copes with this by getting the KSs at different levels to cooperate in the solution process. In doing this, HEARSAY-II combines both top-down and bottom-up processing and reasons about resource allocation with a process called <u>oppurtunistic scheduling</u>. A more detailed description of this concept can be found in [Hayes-Roth, et al, 83].

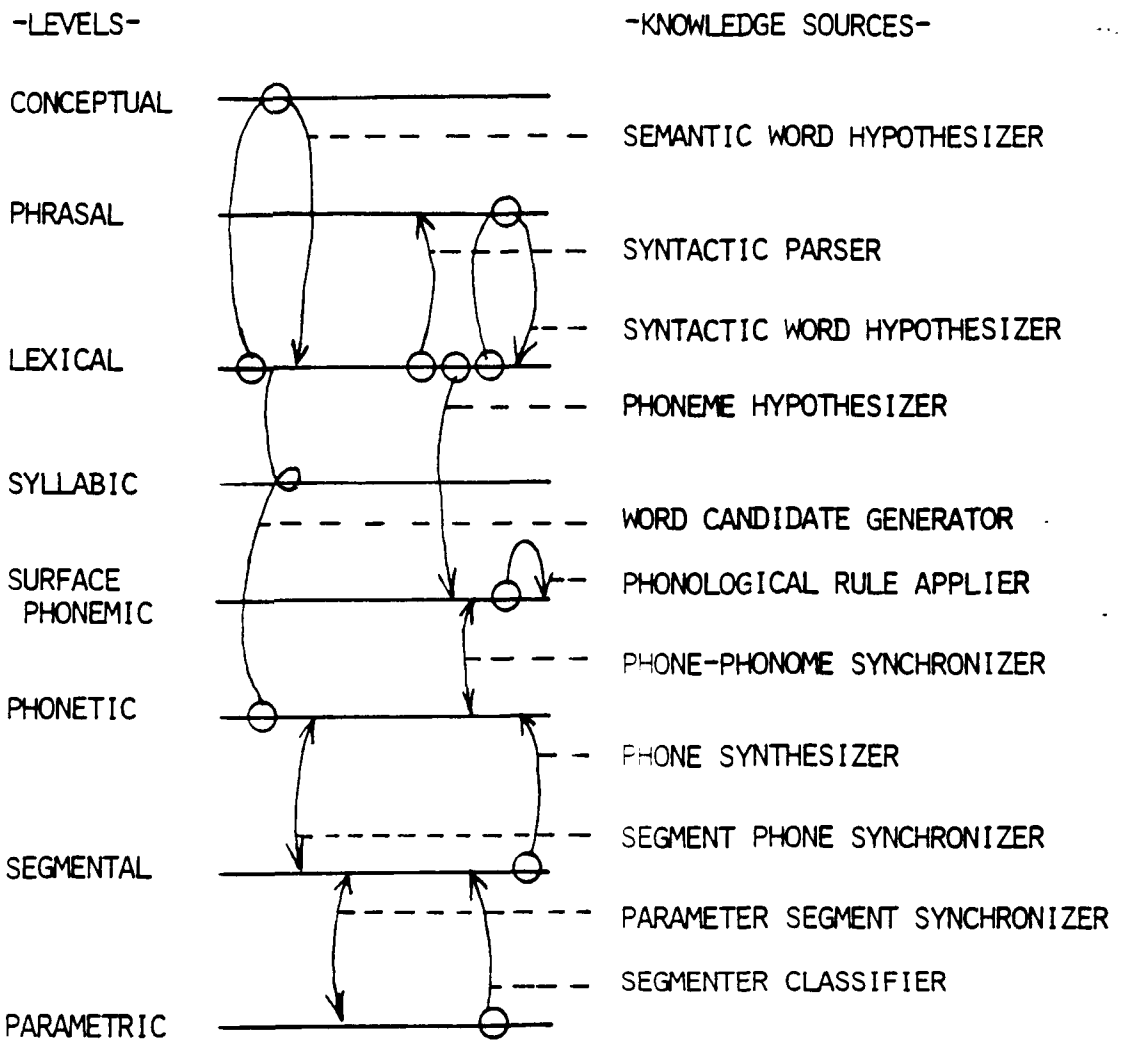


FIGURE 3-28. HEARSAY II LEVELS OF REPRESENTATION AND KNOWLEDGE SOURCES BASED ON [ERMAN, ET AL, '80]

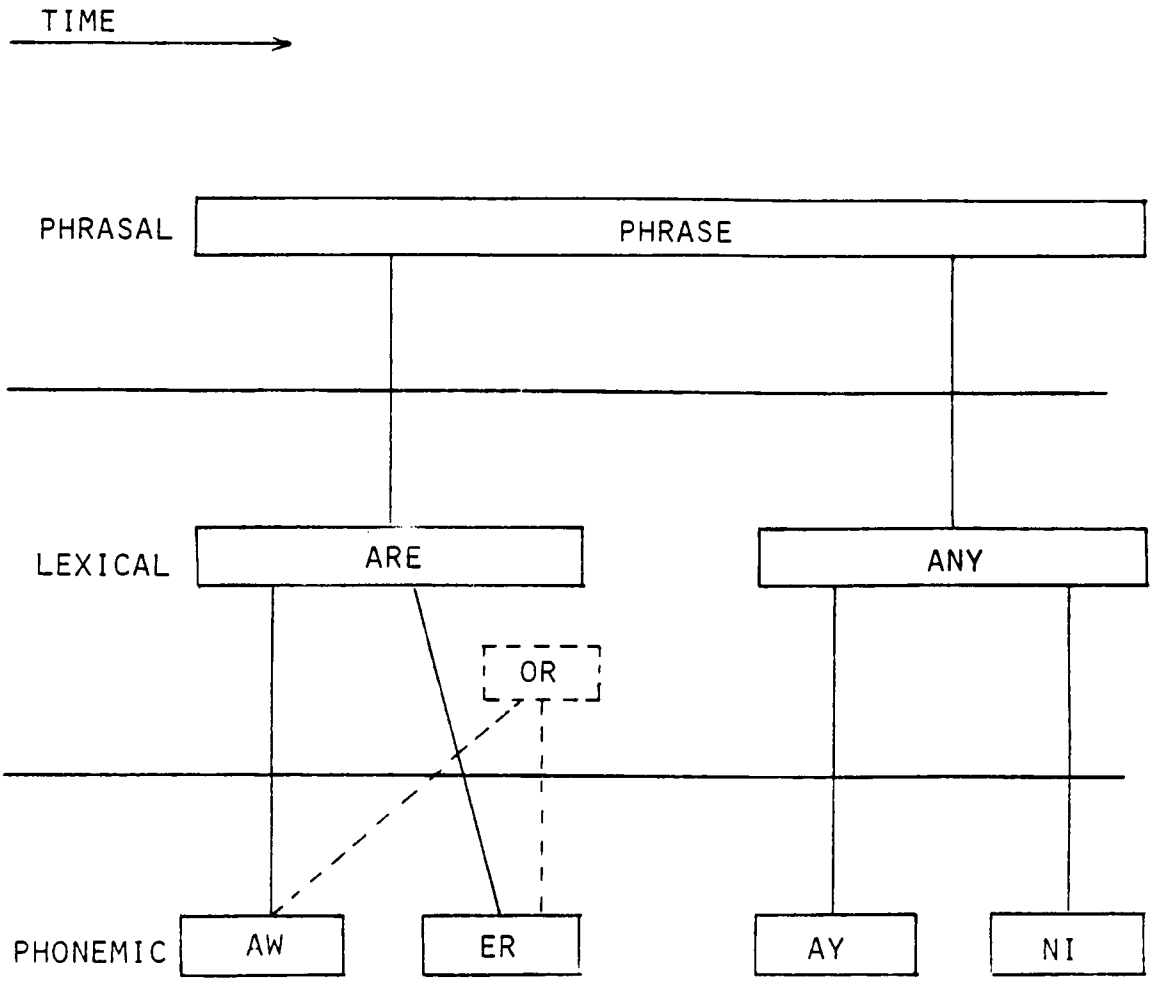


FIGURE 3-29. BLACKBOARD EXAMPLE
BASED ON [ERMAN, ET AL, '81]

C-3

To summarize, a blackboard fills all the roles of a workspace representation: agenda, plan, history, and solution set.

(1) Plan - The blackboard is the globally visible data structure and multiple levels provide the necessary abstractions for searching a large space.

(2) An Agenda - When an hypothesis is placed in the blackboard, it is to be presented to the KSs that have the hypothesis level as their input level, and the set of all such presentations that have not yet been performed on the agenda.

(3) A History - The support represented explicitly in the blackboard is a trace of the evolution of the system's state.

(4) Solution Set - The candidate hypotheses reside at each level in the blackboard along with a label chosen from a set appropriate to that level.

3.4.3 AND/OR Graph

Figure 3-30 shows an example of an AND/OR graph. The example is a formation of a plan to go to New York City. Each node in the graph is subgoal of its parent node (since this is a graph, a node may have more than one parent). The goal "go to New York" can be satisfied by satisfying either the subgoal "Drive a Car" or the subgoal "Fly by Plane", and is, thus, called an OR node. The subgoal "Buy a Plane Ticket" is satisfied by satisfying both the subgoals "Get Money", and "Select an Airline", and, thus, is called an AND node. AND nodes in Figure 3-30 are shown by connecting emanating edges with an ampersand (&). A workspace representation such as this is called an AND/OR graph and is used in many systems with production rule knowledge sources.

As shown in Figure 3-30, the node expansion (for a goal directed graph) continues until a satisfying set of nodes have been generated, all of which are primitive. A primitive node is one that poses a problem that is known to be solvable without a search by the system.

To summarize, AND/OR graphs fill the requirements for a workspace representation:

(1) Plan: The graph presents the global data structure and includes goals and subgoals.

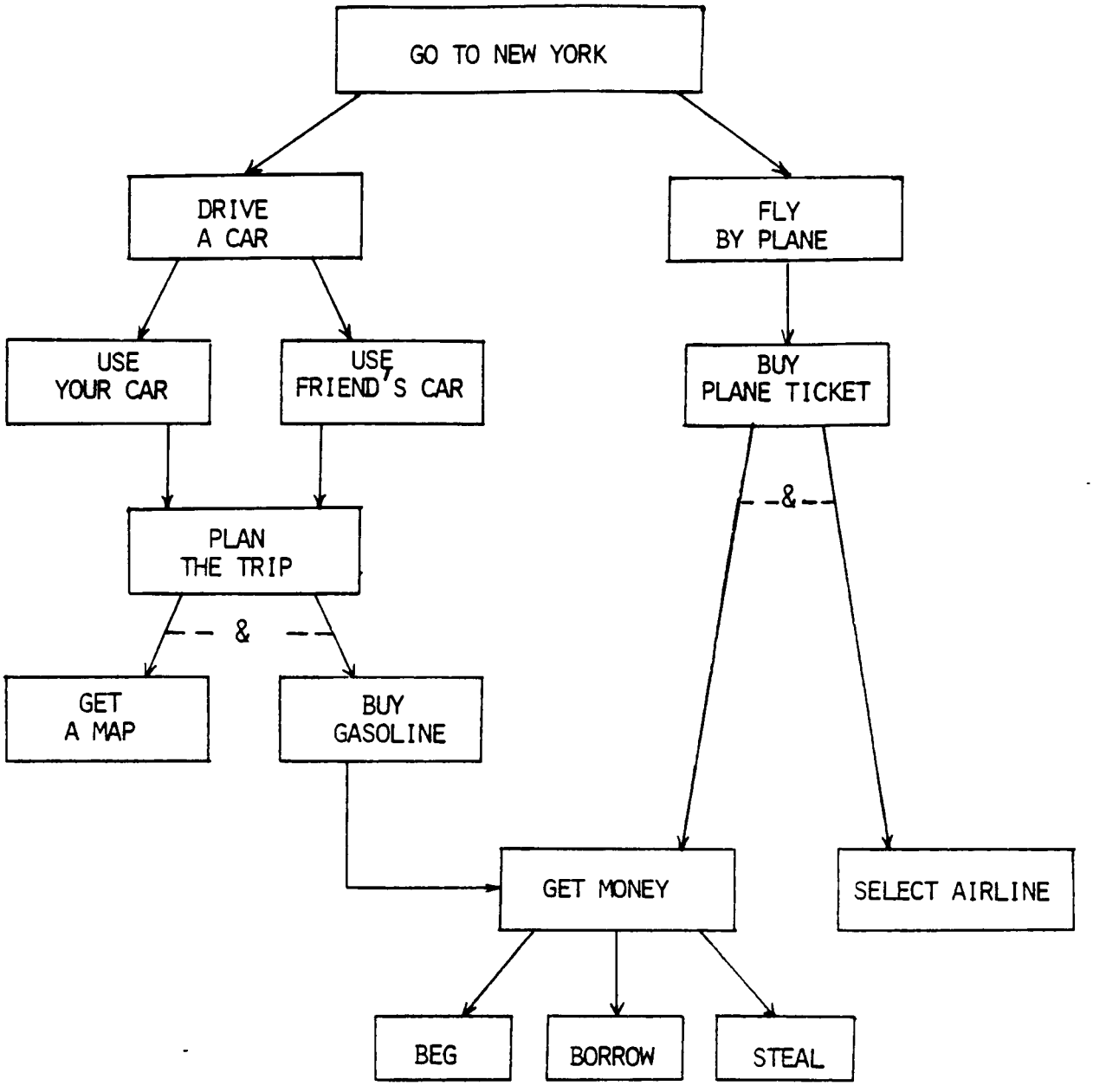


FIGURE 3-30. EXAMPLE AND/OR GRAPH

(2) An Agenda: The agenda is the set of expanded nodes.

(3) A History: The labeled links (not shown in the Figure 3-30) give a reason for the existence of each entity.

(4) Solution Set: Each candidate hypothesis or goal could be represented with an AND/OR graph.

3.4.4 Blackboard Versus AND/OR Graph

By comparing the AND/OR graph with the HEARSAY-Blackboard, one can recognize that:

(1) AND/OR graphs have a more uniform structure that can sometimes be exploited for efficiency.

(2) The HEARSAY-Blackboard has a better structure if the problem decomposes into levels of representation and the system has many knowledge sources.

3.5 The Interface

3.5.1 Functions of the Interface

The interface is the communication port between the KBS and the outside world. Based on the functions provided, the interface of a KBS can be viewed as three different interfaces:

(1) User Interface

(2) Knowledge Acquisition (Expert) Interface

(3) Data Interface.

The user interface provides the necessary facilities for the user as a poser of problem and a consumer of results (answers and justifications or explanations).

The expert interface is the system's port of knowledge acquisition and is used by a domain expert as the provider of knowledge for the knowledge sources (KSs).

The external data interface is similar to that of most other interactive computer systems, in that it incorporates:

(1) Facilities for user input of parameter, data and responses to the system's queries.

(2) The mechanism for locating and accessing files or data bases.

Many of the functions necessary to provide the data interface may be drawn directly from the computer system environment within which the KBS functions, and, thus, they are not discussed here.

In the remainder of this section, User Interface, Expert Interface, and Knowledge Acquisition process are discussed in detail.

3.5.2 User Interface

3.5.2.1 Introduction

The user interface critically affects the acceptance of a KBS by users of the intended domain. The users are (typically) neither computer scientists, nor programmers. A well designed and properly functioning user interface not only minimizes the problems associated with learning any new system, but also, in the long run, improves system productivity by making it possible for the users to be more cooperative in problem solving activities [Barnett & Bernstein, 77]. In short, a good interface could make the difference between a successful KBS and unsuccessful one.

The user interacts with the interface interactively in a jargon specific to the domain of the KBS. The advantage of interactive usage is that the user provides only the necessary information and could request explanations of system behavior and results during problem solving activity.

Besides interacting with a KBS in domain specific jargon, the user inputs the information (and the system outputs results, explanations, etc.) in some restricted variant of English-like natural language. Thus, the user interface acts as a natural language processor. Since handling natural language and all of its complexities is equivalent to solving the entire problem of machine understanding and natural language simulation, only a brief discussion of some techniques will be presented here.

3.5.2.2 User Interface Characteristics

Besides domain specific jargon and English-like natural language, the user interface should possess two additional characteristics: soft-failure and self-knowledge.

(1) Soft-Failure: A KBS should tolerate small or simple errors in a user's input. For instance, if the user's input consists of spelling mistakes, a KBS should not only inform the user, but also guide him as to what are acceptable responses, if not correct the errors itself. An example of this type of spelling corrector is described in [Teitleman, 72].

(2) Self-Knowledge: A KBS system should be able to know what it can and it cannot do. For example, it should be able to answer user's questions like "Can you handle problems about X ?" or "What do you know about Y?" A system with self-knowledge available has the potential to accomodate new users in a reasonable manner [Barnett & Bernstein, 77].

3.5.2.3 The User Input

There are many techniques to implement the input side of the user interface. Parsing is one of the widely used techniques.

Parsing is the process of "picking apart" the sentences that were input to the system and determine their meaning, thus

providing the foundation for providing an appropriate response. There are at least seven different strategies.

3.5.2.4 Parsing Strategies

(a) Backtracking Versus Parallel Processing

Some elements in a natural language do not always have unique meanings. Ambiguities like these force the parser to make choices between multiple alternatives as it proceeds through a sentence. Alternatives may be dealt with all at the same time (called parallel processing), or one at time using a form of backtracking - backing upto a previous choice-point in the computation and trying again. Both these strategies require a significant amount of bookkeeping to keep track of multiple possibilities.

(b) Top Down Versus Bottom Up Processing

This is similar to forward chaining vs. backward chaining as discussed in Section 3.3.4. A parser can operate from the set of possible sentence structures (top down), or from the words actually in the sentence (bottom up).

In a strictly top down approach, a parser begins by looking at the rules for the top level goal structure (sentence, clause, etc.); it then look up rules for the constituents of the top level structure and progresses until a complete sentence structure is built up.

In a strictly bottom up approach, a parser first looks at the rules in the grammar to combine the words of the input sentence into constituents of larger structures (phrases and clauses). These structures will be recombined to show that all input words form a legal sentence in the grammar.

(c) Choosing How to Expand or Combine

In both strategies discussed above, it is necessary to decide how words and constituents will be combined (bottom up) or expanded (top down). There are two basic methods: fixed directionality and variable directionality.

In fixed directionality, the system proceeds systematically in one direction (normally left to right). In variable directionality (also called island driving), the system starts anywhere and systematically looks at neighboring chunks of increasing size (see the discussion in Section 3.3.3.5, "Directionality of Control Strategies").

(d) Multiple Knowledge Sources

In natural language processing systems, particularly in speech understanding systems, another strategy is to arrange knowledge into various levels (phonemic, lexical, syntactic, semantic, etc.), so that the parser can use relevant sets of facts from a variety of knowledge sources (see Section 3.4.2).

3.5.2.5 Parsing Systems

Various natural language processing systems deal with the above seven design issues in different ways. A few selected systems are discussed within this section.

(a) Template Matching

ELIZA [Weizenbaum, 66] is a system of this type. ELIZA (humorously) simulates a Rogerian psychiatrist. Inputs are processed against a series of predefined templates binding the variables of the template to corresponding pieces of the input string. Inputs are matched to patterns like

$1 x(i) {IS/ARE} NOT $2

where $1 matches any string of words and x(i) matches any single word. Responses are built up by giving corresponding output patterns such as

WHAT IF x(i) WERE $(2) ?

Given the input "Today's temperature is not hot", the system could produce the response, "What if temperature were hot?" This is accomplished by matching $1 to "Today's", x(1) to "temperature", and $2 to "hot".

ELIZA and other systems (like SIR and STUDENT) using this kind of matching techniques were successful as long as the domain and style of dialog is sufficiently constrained and the system's

designer could incorporate appropriate templates. However, the method was inextensible, and template matching was soon abandoned in favor of more sophisticated techniques [Barr & Feigenbaum, 81].

(b) Transition Networks

Perhaps the best known and widely used technique for parsing is the augmented transition network (ATN). ATNs were first developed by Woods [Woods, 73]. The concept of an ATN evolved from that of a finite state transition diagram, with the addition of tests and "side effect" actions to each arc.

Figure 3-31 shows a finite state transition diagram (FSTD). Boxes with S and E represent the initial and final states, respectively. The FSTD accepts any phrase that begins with "the", and ends with a noun and has an arbitrary number of adjectives in between. For example the FSTD shown in the Figure 3-31 accepts the input phrase "the pretty picture".

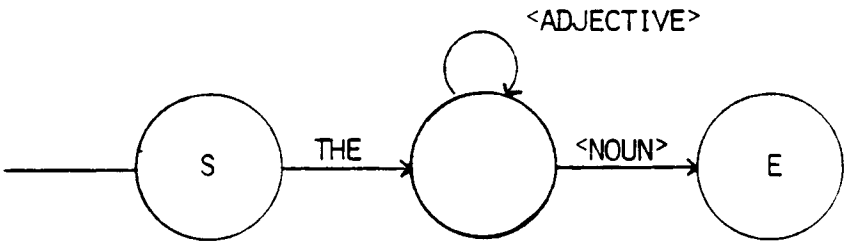


Figure 3-31. A Finite State Transition Diagram

(i) Recursive Transition Networks

Grammars like the ones shown in the Figure 3-31 are inadequate for dealing with the complexity of natural language representation. To increase the power of recognition, FSTD can be extended in a natural way to include recursion mechanisms. These extended FSTDs are called recursive transition networks (RTNS).

Figure 3-32 shows an RTN (taken from [Barr & Feigenbaum, 81]). In this figure, NP denotes a noun phrase; PP a prepositional phrase; det, a determiner; prep, a preposition; and adj, an adjective. If the input string is "The little boy in the swimsuit kicked the red ball", the above network would parse it into the following phrases:

NP: The little boy in the swimsuit

PP: in the swimsuit

NP: the swimsuit

Verb: kicked

NP: the red ball

In Figure 3-32, one can notice that any subnetwork of an RTN may call any other subnetwork, including itself. One can also notice that an RTN may be non-deterministic in nature; that is, there may be more than one possible arc to be followed at a given point in a parse. These alternatives can be handled either by parallel processing or by backtracking, as discussed in Section 3.5.2.3.

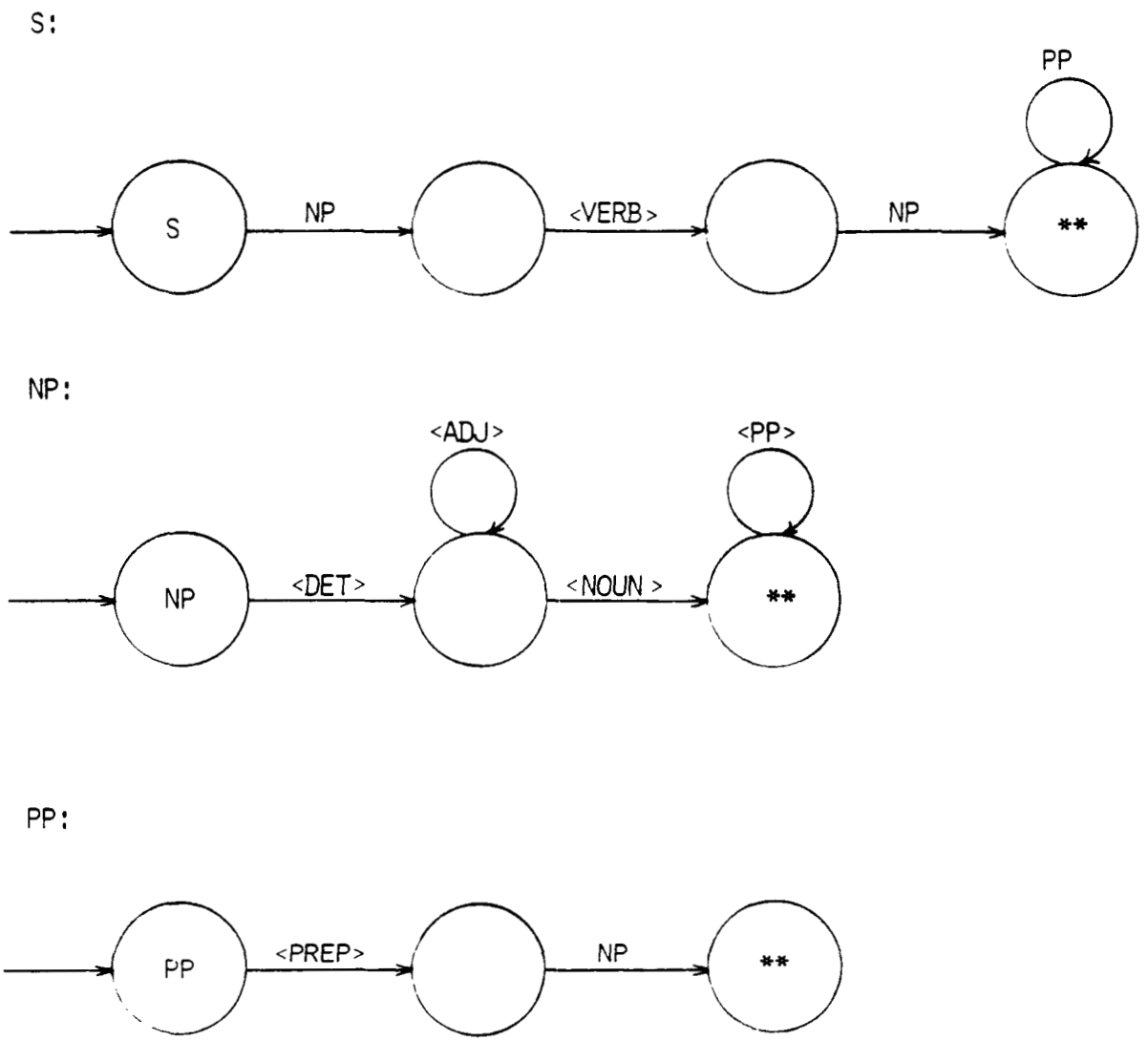


FIGURE 3-32. A RECURSIVE TRANSITION NETWORK
BASED ON [BARR & FEIGENBAUM, '81]

(ii) **Augmented Transition Networks**

An ATN is an RTN that has been extended in three ways:

(1) A set of registers has been added; these can be used to store information, such as partially formed derivation trees (like the two shown in the Figure 3-32), between jumps to different networks.

(2) Arcs, aside from being labeled by word classes or syntactic constructs, can have arbitrary tests associated with them that must be satisfied before the arc is taken. This makes it possible to enforce such constraints as verb agreement with the subject: for example, accept "he goes" but not "he go".

(3) Certain actions may be "attached" to an arc, to be executed whenever it is taken (usually to modify the data structure returned).

ATNs have been successfully used in question answering systems (LUNAR) [Woods, 73b], text generation systems (SHRDLU) [Winograd, 72], and speech understanding systems (HWIM) [Wolf & Woods, 80].

One limitation of the ATN approach is that the heavy dependence on syntax restricts the ability to handle ungrammatical (although meaningful) utterences.

(c) Semantic Grammar Parsers

There is another class of methods for understanding natural language which do not use any explicit syntax, but rather depend on a semantic abstraction of the problem domain. For instance, such a semantic grammar for a system that talks about airline reservations could have grammatical classes like ‹DESTINATION›, ‹FLIGHT›, ‹FLIGHT-TIME›, and so on. This abstraction of concepts along with knowledge of English key words (e.g., of) forms a parser. This kind of technology has the advantage of being efficient and easy to use in a variety of domains. It works well as long as the domain is reasonably bounded (like a front end to a KBS) but would not be extensible to more unrestricted areas.

The LIFER [Hendrix, 77] and SOPHIE [Brown, et al, 83] systems use semantic grammar based parsers.

3.5.2.6 Output to the User

The other half of the user interface is responsible for output generation. This part is responsible for (a) accepting the input from the user, (b) providing explanations and results during a problem solving activity, and (c) answering questions about the system itself. Of all these tasks, providing explanations is the most difficult. This is because:

(1) The explanation must be in terms of the knowledge chunks, problem parameters, and inference rules that

were used to derive the results;

(2) The internal representation must be translated to a format suited for human understanding.

As was discussed in Section 3.1.5, the ability of a KBS to provide good explanations depend on the chunk size. If the knowledge chunks used are too small, the explanation is laborious and may not be satisfactory; on the other hand, if the chunks are too large the explanation mechanism may be unnatural to the user. Similarly, the ability to provide good explanations depends on the selection of relevant or crucial inference rules for solving the problem at hand (unless asked for additional details, in which case the system should respond appropriately).

3.5.2.7 Methods of Providing Explanations

(a) Workspace Representation

As was discussed in Section 3.4, a workspace representation offers a straightforward method for providing explanation. A workspace representation stores the history of the problem solving activity. The elements in a workspace representation are associated with the rule of inference and what rule was applied on other workspace elements, knowledge chunks, confidence factors, etc.

The explanation mechanism can start from the element(s) of the workspace representing the problem solution and pick out the

sequence of events that moved the system from problem definition to solution. The advantage of this approach is that the explanation mechanism could use all the useful information stored in a workspace including why a particular solution was selected and why others were rejected. The disadvantage is that most of the information may never be used.

(b) Using Knowledge Source(s)

In this method, the KS determines the most relevant information for an explanation and a knowledge chunk can optionally have an explanation scheme. During a problem solving activity, if a knowledge chunk is used, the scheme (associated with that chunk) is instantiated in its local environment to produce an explanation. The advantages of this approach are:

(1) High-quality explanations can be produced because it is possible to take idiosyncratic situations into account.

(2) The explanation mechanism can be used for other purposes, for example, part of the complaint department for a frame (see Section 3.2.7 "Frames").

The disadvantage of this method is that the expert who provides knowledge to the system must consider the method and necessity of explaining each knowledge chunk.

(c) Re-solve the Problem

In this method, a problem is solved without keeping a history in the workspace. If the user asks for an explanation, the method must re-solve the problem in a careful mode i.e., the explanation mechanism carefully watches the inference engine during its re-solving activity and selects the events that are of likely interest. This is done by attaching a set of special demons (see Section 3.2.3) that are triggered when special situations occur. At these points, the explanation mechanism can interrupt normal processing to perform the necessary data collection.

The advantage of this method is a possible gain of efficiency if explanations are rarely requested. The disadvantage of this method is the inefficiency introduced into the inference engine so that demon-like execution could occur.

3.5.3 Expert Interface

3.5.3.1 Introduction

Expert interface is used by a domain expert, the provider of knowledge for a knowledge base and the system implementors (or knowledge engineers) who are responsible for building the initial knowledge base (this interface is also called the knowledge acquisition interface). Because of this, one can assume that the user of the expert interface has some knowledge and awareness of the structure and functions of the KBS. This, of course, does

not imply that the expert is a programmer; rather it means that he basically knows how knowledge is represented (for example, by IF-THEN production rules) or how uncertainty of knowledge is handled (for example, by certainty factors).

3.5.3.2 Expert Interface Tasks

The expert interface (or knowledge acquisition interface) has three major tasks [Barnett & Bernstein, 77]:

(1) Accepting knowledge in external format and translating it into internal format.
(2) Validating the consistency of new and old knowledge.
(3) Storing the knowledge into the KB.

This three step process is called compilation.

The first task is usually handled by using a part of the input mechanism from the user interface which can handle restricted natural language.

The second task is a more difficult one. This involves validation of consistency, and checking for redundancy, a task complicated by the presence of confidence (or credibility) factors.

Redundancy can be checked by proving that new knowledge can be derived from the existing knowledge base. Inconsistency can be checked by adding the new knowledge to the old knowledge and proving something that is patently false, say A & ~A; if there is

no inconsistency, the proof will fail; otherwise, the proof will succeed. For more detailed accounts of the problems of maintaining consistency, see [McDermott, 74].

The third task, storing the new knowledge into the KB is called accommodation. This task becomes more difficult if a system has several knowledge sources and fact files in the KB.

Storing is a very complex process. This is because the internal (physical) representation is usually a structure with links between chunks, and the acquisition mechanism must insert the new chunk into this complex network.

For example, in MYCIN, each production rule that concludes something about feature F is linked to every rule that tests F in its antecedent (left hand side). Thus, the insertion (as well as deletion and modification) of knowledge chunks is a complex operation that involves many things such as confidence factors, conflict resolution strategies, existing knowledge base contents, etc.

In the next section, the knowledge acquistion process, which is a major bottleneck in developing KBSs, is described.

3.5.4 Knowledge Acquisition (KA) Process

3.5.4.1 Introduction

The stages involved in the KA process can be characterized as problem identification, conceptualization, formalization,

implementation, and testing, as shown in Figure 3-33. In reality, KA may not be as neat and well defined as the figure suggests.

3.5.4.2 Problem Identification

This stage is further divided and discussed below.

(a) Participant Identification and Roles

The first thing that should be done before the KA process can begin is the selection of participants, and definition of their roles. This could mean the selection of a domain expert and a single knowledge engineer. The KA process can also include other participants: multiple domain experts, multiple knowledge engineers, and even interdisciplinary experts.

(b) Problem Identification

The objective during this phase is to characterize the problem and its supporting knowledge structures so that the development of the KB can begin. Many iterations may be needed during this phase because a knowledge engineer and/or domain expert may find that the initial problem considered is too large or unwieldy for the resources available. At the end of this phase, both the knowledge engineer and the domain expert must arrive at a final, informal description they can agree on for the problem identification.

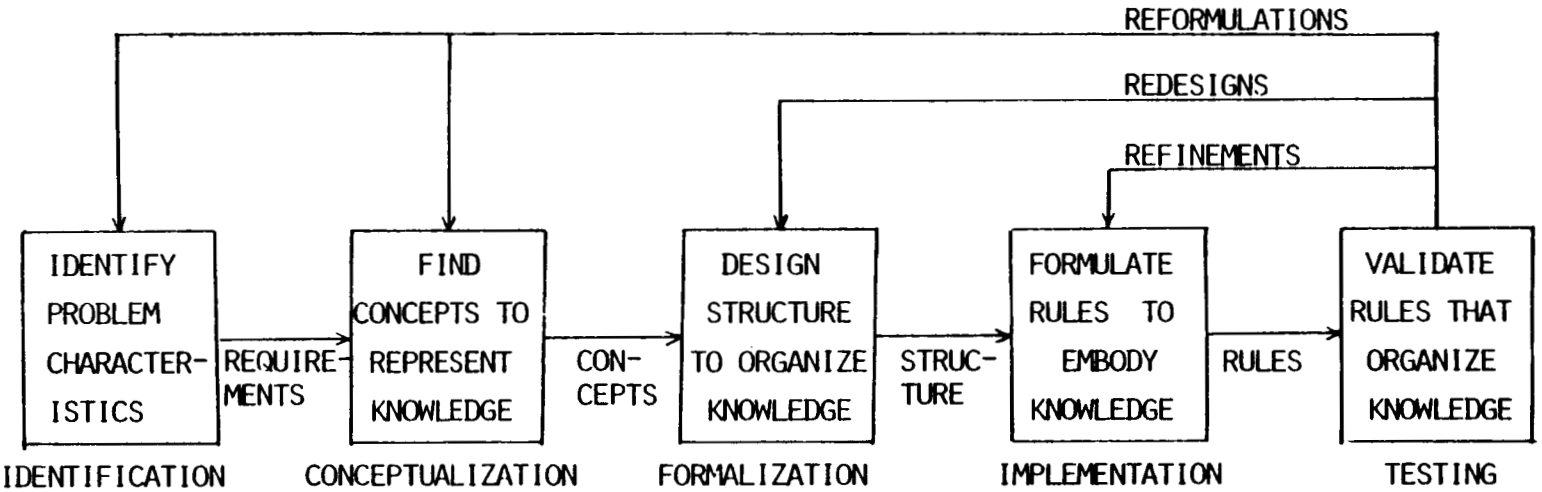


FIGURE 3-33. STAGES OF KNOWLEDGE ACQUISITION
BASED ON [HAYES-ROTH, ET AL, '83]

(c) Resource Identification

During this phase, the knowledge engineer and domain expert must use various sources to obtain knowledge relevant to building the KBS. For the domain expert, these include textbooks, journals, past problem solving experience, etc. For the knowledge engineer the sources include experience on analogous problems and knowledge about methods, representations, and tools for building KBSs.

3.5.4.3 Conceptualization Stage

During this stage, key concepts and relations (that were mentioned during the identification stage) are made explicit. The knowledge engineer represents these concepts and relations in a diagram that serves as a base for the prototype system. Some of the questions that need to be answered before proceeding with the conceptualization process are:

(1) What data types are available?

(2) What is given and what is inferred?

(3) How are the objects in the domain related?

(4) Can you diagram a hierarchy and label causal relations, set inclusion, part or whole relations, etc.? What does it look like?

(5) Can you identify and separate the knowledge needed for solving a problem from the knowledge used to justify a solution?

This stage also involves repeated interactions between the knowledge engineer and the domain expert.

3.5.4.4 Formalization Stage

This stage involves mapping the key concepts, subproblems, and information flow characteristics isolated during conceptualization into more formal representations based on various knowledge engineering tools and languages.

During this phase, the knowledge engineer takes an active role, telling the domain expert about the existing tools, representations, and problem types that seem to match the problem at hand. During this phase, the knowledge engineer must evaluate the disadvantages of mismatches that will occur when a single tool is chosen and select the one with the least overall disadvantages. At the end of this phase, a set of partial specifications describing how the problem can be represented within the chosen tool or framework should be completed.

3.5.4.5 Implementation Stage

The domain knowledge made explicit during the formalization stage specifies the contents of the data structures, the inference rules, and the control strategies. The tool or representation framework chosen specifies their form. Thus the implementation phase involves mapping this formalized knowledge into the representational framework associated with the tool

chosen, i.e., the implementation phase involves the development of a prototype KBS.

The prototype KBS is implemented by using whatever knowledge engineering aids are available for the chosen representation (for example, a knowledge base editor). If the existing tools are inadequate and/or inappropriate, it may be necessary to develop new tools.

3.5.4.6 Testing Stage

This stage involves evaluating the prototype system and the representational forms used to implement it. Once the system performs well with two or three examples, it should be tested with a variety of complex examples to determine the weaknesses in the knowledge base and inference structure. These weaknesses should be corrected, and a revised prototype should be developed. This may involve repeating one or more of the phases discussed above.

For a more detailed discussion on the knowledge acquistion process, see [Hayes-Roth, et al, 83], on which the above discussion is based.

3.5.4.7 Difficulties in Knowledge Acquisition

There are many major difficulties in acquiring knowledge for a KBS:

(1) One of the most outstanding problems is the representational mismatch, the difference between the way a human expert expresses knowledge and the way it must be represented in the KBS environment. In recent years, researchers have focussed on developing KA tools that could decrease the representational mismatch. One method used in some systems (example: ROSIE [Fain, et al, 81]) to decrease this mismatch is to allow the expert to converse with the system in natural language.

(2) Another major difficulty in KA is verbalization by the expert. It is almost always difficult for the human expert to describe the knowledge in a formal way. Therefore, in order to build a KBS it is necessary for the expert to rethink his methods and procedures. One method for starting this process is the protocol study.

(A protocol study is a process in which an expert is given a problem to solve, and the knowledge engineer observes and/or records the expert's behavior or asks for explanations of various steps. The knowledge engineer then analyzes the collected information and tries to determine general patterns, knowledge used, and principles of reasoning.)

(3) Other major difficulties in the KA process result because of limitations on current technology. Representation languages and tools used by current systems are limited in their expressive capabilities. Similarly, techniques to allow systems to be gracefully extended are very limited.

The above mentioned problems - mismatch, formalization, expression, and extendability - all contribute to what is known as the knowledge acquisition bottleneck. Currently, this is one of the very active areas of research in building expert systems.

Chapter 4

KBS BUILDING TOOLS AND LANGUAGES

4.1 Introduction

At the moment, construction of KBSs and experimentation with them are both very expensive and time consuming. Recognizing this, researchers have recently begun developing programming languages and tools for building KBSs. While these tools and languages are just coming into use and are certain to undergo further development, they promise to reduce significantly the programming effort needed to develop a new system as well as modify it [Duda & Gashing, 81].

These languages and tools can be categorized into four different groups (based on [Hayes-Roth, et al, 83]):

(1) General purpose programming languages.

(2) Skeletal systems.

(3) General purpose representation languages.

(4) Computer-aided design tools for KBSs.

The discussion in this chapter is generally based on [Hayes-Roth, et al, 83].

4.1.1 General Purpose Programming Languages

Some AI programming languages have very powerful features and can be used to implement a system from "scratch". LISP, developed by McCarthy in 1958, is chosen for much work in AI. LISP has some advanced features like: symbol manipulation, list processing, and recursion. These features provide a high level conception of data and control. In addition, the programmer can be freed from certain burdens (like how to manage memory) that could slow down the experimental process.

There are at least six other AI languages, that have been developed during the past two decades:

PLANNER [Hewitt, 71]

CONNIVER [Sussman, et al, 72]

QLISP [Green, 69]

SAIL [Feldman, et al, 72]

POP-2 [Popplestone, 67]

FUZZY [Le Faivre, 77]

Except for LISP, none of these languages are in widespread use. There are two commonly used LISP dialects, INTERLISP, developed at BBN and XEROX [Teitelman, 78], and MACLISP, developed at MIT. The choice of one of them "is probably more a matter of personal preference and availability than of clear technical superiority", although advocates of MACLISP and INTERLISP often seem to be claiming that superiority [Hayes-Roth,

et al, 83].

As was discussed earlier, the two most important components in a KBS are the inference engine and the knowledge base (or a set of rules). Any language which is chosen for construction of a KBS should provide facilities for both.

Let us first consider the representation of a knowledge base (a body of rules). Depending on the general framework, each rule should satisfy a set of conditions (which are relevant) and perform a set of actions (when invoked). For example, consider the following statement or informal rule (refer the example in Section 3.2.5)

"Low fan belt tension causes alternator output to be low."

This statement can be represented as

```
(IF (CAUSE BELT_TENSION LOW)
 THEN (CONSEQUENCE ALTERNATOR_OUTPUT LOW)
 )
```

The above rule can be represented more generally and formally in a Backus-Naur form (BNF) as follows:

```
‹rule› ::= (IF {‹antecedent›} THEN {‹consequent›})
‹antecedent› ::= ‹associative triple›
‹consequent› ::= ‹associative triple›
‹associative triple› ::= (‹attribute› ‹object› ‹value›)
```

where ‹attribute›, ‹object›, and ‹value› would be domain specific terms. Using this type of formal rule language, a knowledge base (or body of rules) can be constructed.

Now let us consider the second aspect: the inference engine. Ideally the same IE could be used for various domains, by just changing the rule set. For example, the following is a simple backward chaining inference engine (discussed in detail in Section 3.3.3) for the rule language given above.

```
To test whether hypothesis X is true:

if X is stored in the global data base
then X is true
else if there are any rules whose consequents
include X
     then for each such rule:
     if all antecedents are true
     then add all consequents to the global data base
     and X is true
     else if the user says that X is true
     then X is true
     else X is false.
```

Note how back-chaining is implemented above. Checking the antecedents of a rule causes the inference engine to be invoked recursively.

The above example and discussion is provided to give a

flavor of AI languages and no attempt is made to describe them in detail. A thorough introduction to some AI programming language features can be found in the excellent book "Artificial Intelligence Programming" by Chairniak, Riesbeck, and McDermott [Chairniak, et al, 79].

4.1.2 Skeletal Systems

EMYCIN, EXPERT, and KAS are examples of this category. In these systems, domain specific knowledge is explicitly represented as rules in a KB, rather than coding in an inference engine. This clear separation of the KB and the IE permits the KB (or domain specific rules in the KB) to be replaced with another KB (with different domain specific rules).

For example, EMYCIN (for Essential MYCIN) is the MYCIN system without the medical knowledge (specialized knowledge of meningitis as well as some general knowledge about medicine). Using EMYCIN, two experimental systems were developed: PUFF [Fagan, et al,, 79] and SACON [Bennet & Englemore, 79].

PUFF was built by replacing MYCIN's infectious disease rules by rules for pulmonary function diagnosis and SACON was built for psycho-pharmacology.

Even though the above mentioned systems are reported to be successful, building "general systems" - systems that can be applied to another domain merely by removing the rules for a given domain and substituting rules for the new one - is, in

practice, not that simple. The following are among the problems that may occur [Hayes-Roth, et al, 83]:

(1) The old framework may be inappropriate to the new task. This is both the most likely and most serious problem.

(2) The control structure embodied in the IE may not sufficiently match the new expert's way of solving problems.

(3) The old rule language may be inappropriate to the new task.

(4) There may be task specific knowledge hidden in the old system in unrecognized ways.

4.1.3 General Purpose Representation Languages

OPS5, HEARSAY-III, RLL, and ROSIE fall into this category. These tools (or languages) are less constrained than skeletal systems, since they are not as closely tied to a particular framework. Thus, they allow for a wider variety of control structures and can be applied to a broader range of tasks, though the process of applying them may be more difficult than with skeletal systems.

For example, OPS5 [Forgy, 80], incorporates a general control and representation mechanism and it is not biased towards

a particular problem solving strategies or representation schemes. OPS5 has been used for a variety of applications in the area of AI and cognitive psychology, as well as building R1, the expert system for configuring VAX computers [McDermott, 80].

In addition, OPS5 provides other facilities: the OPS5 interpreter provides the programmer with a conventional interactive programming environment much like that of a typical LISP interpreter - to trace and break runs, to examine the state of the system, to change the system in the middle of a run, and so on.

4.1.4 Computer Aided Design Tools for Building KBSs

AGE [Nii & Aiello, 79] falls into this category. Specifically designed to allow the implementation of broader spectrum of KBs, AGE gives the designer (user) a set of a separate, interconnectable preprogrammed modules for selecting a framework, implementing the KB, IE, and the data base. Thus AGE differs from other skeletal systems in one important dimension: it provides an environment in which the designer can choose or specify a variety of knowledge representations and processing methods. For example, an AGE user is able to build and run a program that behaves in ways similar to a program built using EMYCIN or one built using HEARSAY-III. AGE also contains knowledge about its own facilities, procedures, a tutor subset (that lets the user browse online manual), and a design subset

(that provides online advise on the AGE itself).

4.2 Case Studies

This section presents detailed description of three tools (or languages) mentioned in the previous section. They are:

EMYCIN (skeletal system)
HEARSAY-III (general purpose representation language)
AGE (computer aided design tool)

The discussion is primarily based on [Hayes-Roth, et al, 83], and the references identified with the respective systems.

4.2.1 EMYCIN

4.2.1.1 Overview of EMYCIN

EMYCIN is basically a domain-independent version of MYCIN i.e., a MYCIN system without the medical knowledge. EMYCIN is a skeletal system for developing a consultation program that can request data about a case and provide an interpretation or analysis. It is particularly well suited to deductive problems such as fault diagnosis, in which a large body of potentially unreliable input data (symptoms, laboratory tests) is available and the solution space of possible diagnoses can be enumerated.

EMYCIN helps a designer build a new KB, and thus a new KBS. The problem specific knowledge can be represented in MYCIN-like rule language and EMYCIN allows the MYCIN inference engine to be

applied to a new KBS. This provides the new KBS with MYCIN's versatile explanation facility.

In addition to these, the EMYCIN system contains a KB editor to aid in debugging an emerging KB. All of the components are shown schematically in Figure 4-1 (from [Buchanan & Duda, 83]).

4.2.1.2 Knowledge Representation in EMYCIN

The knowledge in EMYCIN is represented as production rules (see Section 3.2.5, "Production Rules") in the following rule language:

```
rule ::= (IF ‹antecedent› THEN ‹action› (ELSE ‹action›))
‹antecedent› ::= (AND {‹condition›})
‹condition› ::= (OR {‹condition›}) |
                (‹predicate› ‹associative-triple›)
‹associative-triple› ::= (‹attribute› ‹object› ‹value›)
‹action› ::= ({‹consequent›} | {‹procedure›}
‹consequent› ::= {‹associative-triple› ‹certainty-factor›}
```

A rule links an antecedent to one action if the antecedent is true, and (optionally) to another, if the antecedent is false. The antecedent is always the conjunction of one or more conditions. A condition is either

(1) The disjunction of one or more conditions or

(2) A predicate applied to an attribute-object-value triple (predicate can include negation).

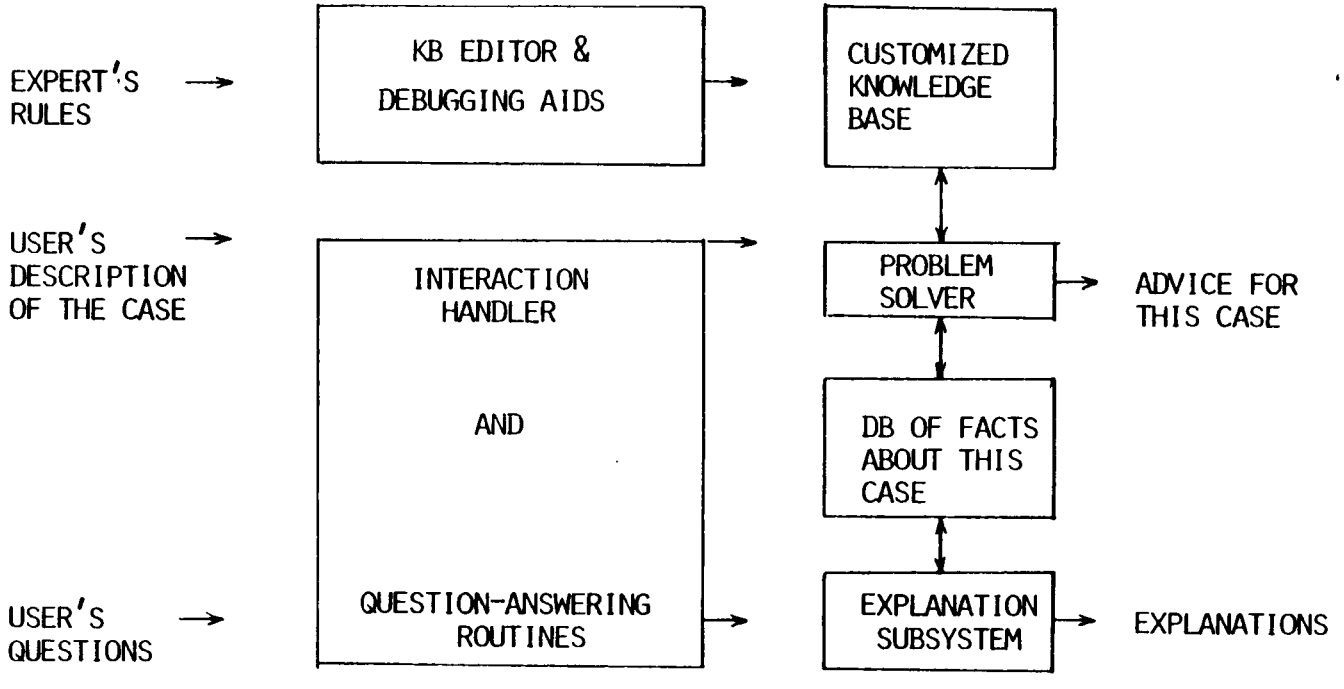


FIGURE 4-1. EMYCIN SYSTEM
BASED ON [BUCHANAN & DUDA, '83]

Thus, the antecedent is an arbitrary Boolean combination of predicates of associative triples.

For example, one of the MYCIN's bacterial infection rule is:

```
IF    ($AND (SAME (CNTXT INFECT PRIMARY-BACTEREMIA)
             (BEMBF (CNTXT SITE STERILE-SITES)
             (SAME (CNTXT PORTAL GI)))
THEN  (CONCLUDE (CNTXT IDENT BACTEROIDS TALLY .7))
```

In English, the antecedent of the rule (everything between IF and THEN) is true if and only if:

(1) The infection is primary-bacteremia, and

(2) The site of culture is one of the sterile sites, and

(3) The suspected portal of entry of the organism is the gastrointestinal (GI) tract.

The objects in the associative triples (called "context" in the EMYCIN terminology) are variables corresponding to domain entities. They are organized into a simple hierarchy called a context tree (Figure 4-2). This serves several purposes:

(1) Binding of free variables in a rule are established by the context in which the rule is invoked with the standard access to contexts which are its ancestors.

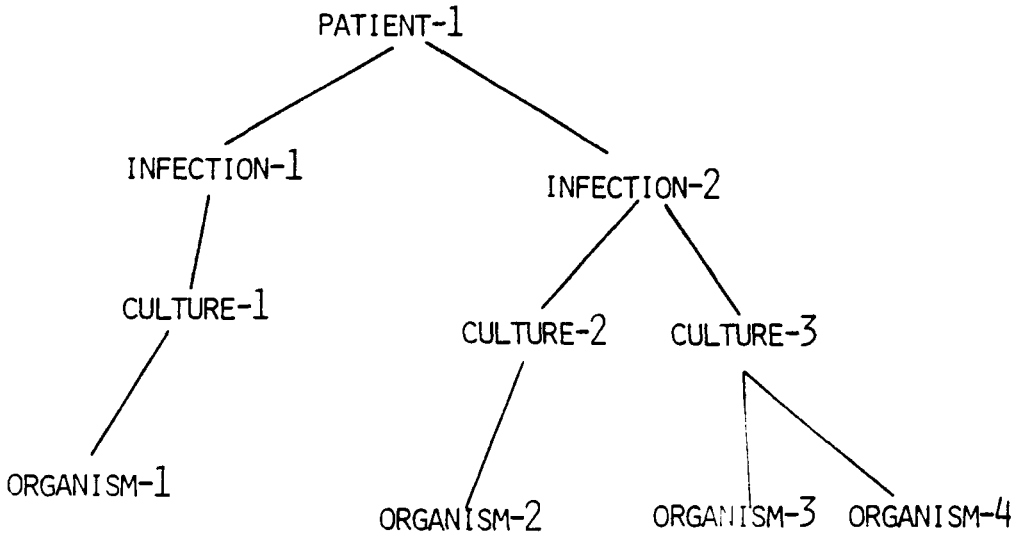


FIGURE 4-2. A SAMPLE CONTEXT TREE
BASED ON [DAVIS, ET AL, '81]

(2) Since this tree is intended to reflect the relationships of objects in the domain, it helps structure the consultation in ways familiar to the user.

For example, in the MYCIN domain, objects might be patient-1, infection-1, infection-2, culture-1, culture-2, culture-3, organism-1, organism-2, etc. The context tree (Figure 4-2) would indicate that ORGANISMS belong to CULTURES, CULTURES belong to INFECTIONS, and INFECTIONS belong to PATIENTS. Thus a context tree provides some of the inheritance mechanisms of a frame representation.

To accommodate uncertainty, EMYCIN associates a certainty factor (see Section 3.1.6) with every attribute-object-value triple. This number ranges from -1 (when the triple represents a false assertion) through 0 (no opinion) to +1 (the assertion is known to be true). Predicate such as SAME can either evaluate to T (true) or some certainty interval (such as 0.2 to 1) or can be fuzzy-set functions that indicate a degree of truth. As in fuzzy-set theory, AND returns the minimum and OR returns the maximum of the certainty values to its arguments [Zadeh, 75]. A rule is considered "true" only when the final certainty is greater than some threshold (typically 0.2), and will be treated as "false" if its final certainty is less than another threshold (typically -0.2).

The action part of a rule either updates (modifies) the

certainty of the specified consequents or evaluates a set of attached procedures. In modifying the certainty, the system combines:

(1) The certainty of the antecedent.

(2) The present certainty of consequent.

(3) The certainty factor associated with the rule according to the CF formulas of Shortliffe and Buchanan [Shortliffe & Buchanan, 75].

4.2.1.3 The EMYCIN Inference Engine

EMYCIN uses backward chaining as a control strategy. Its initial goal is to determine the value of a top level goal attribute. Subsequently, EMYCIN works on the goal of establishing the value of the attribute of some object. This process continues with a precomputed rule set (whose consequents are known to bear on that goal) until either the value is established with complete certainty or exhausts the rule set. If no value can be deduced, it resorts to asking the user for the value.

To apply (or execute) a rule, EMYCIN must first establish the truth of its antecedent, which requires determining the certainty of each of its conditions. To determine the certainty of each of its conditions, the system (typically) has to establish the value of other attributes of objects. This means that the system sets up subgoals that are addressed by using the

same mechanism recursively.

4.2.1.4 EMYCIN Facilities

One of the major benefits of using EMYCIN to build other MYCIN-like systems is its (EMYCIN's) explanation facilities. It allows a user to examine both the reasons for the conclusions reached in a particular session, and its rule set in the knowledge base. This can be done by simple commands like "WHY" and "HOW".

In addition, as already mentioned, EMYCIN has a knowledge base editor. The KB editor checks syntactic correctness of the new rules entered and sees that they do not contradict or subsume existing rules. A contradiction occurs when two rules with the same antecedents have conflicting consequents; subsumption occurs when the antecedent of one rule is a subset of that of another and their consequents are the same [Hayes-Roth, et al, 83].

EMYCIN also provides valuable tracing and debugging facilities. And, finally, libraries of test cases can also be maintained.

4.2.2 HEARSAY-III

4.2.2.1 Overview of HEARSAY-III

HEARSAY-III is a general purpose knowledge representation tool. It also provides a domain-independent framework for

building KBSs. The architecture of HEARSAY-III is based on the architecture of HEARSAY-I [Reddy, et al, 73], and HEARSAY-II [Erman, et al, 80], which are speech understanding systems developed at Carnegie-Mellon University under a DARPA project. Specifically, HEARSAY-III uses HEARSAY-II's concepts of modular knowledge sources and "blackboard" (which provides system wide communication, see Section 3.4.2). However, HEARSAY-III is specifically not a speech understanding system.

The design goals for HEARSAY-III were to develop representation and control facilities with which a user could construct and experiment with a KBS for a chosen domain. Some salient features of HEARSAY-II are:

(1) It supports codification of diverse sources of knowledge. HEARSAY-III is not restricted to any particular class of application domains, and in fact, supports various types of knowledge from various application domains.

(2) It supports application of these diverse sources of knowledge. HEARSAY-III allows flexible coordination of knowledge sources during a problem solving activity.

(3) It represents and manipulates competing solutions that could be constructed incrementally.

(4) It reasons about partial solutions, i.e., HEARSAY-III supports the ability to reason and manipulate the solutions during various stages of their construction.

(5) It applies domain dependent consistency constraints to the competing partial solutions, which results in reducing the search space.

(6) It supports long-term, large-system development. In particular, HEARSAY-III supports experimentation with varying knowledge for the application domain and varying schemes for applying that knowledge.

4.2.2.2 Knowledge Representation in HEARSAY-III

(a) The Underlying Relational Data Base

HEARSAY-III consists of a relational data base system and its corresponding control facilities. The data base language is called AP3 [Goldman, 78], and is embedded in INTERLISP. An AP3 data base provides strong typing on assertion, retrieval, and parameter passing in function calls which can be used by a user of HEARSAY-III for modeling in a specific domain. The HEARSAY blackboard and all publicly accessible HEARSAY-III data structures are represented in the AP3 data base.

AP3 also makes available to HEARSAY-III applications a context mechanism. This context mechanism allows reasoning along independent paths, which may arise both from a choice among

several competing KSs and from a choice among several competing partial solutions.

Finally, AP3 also provides facilities for a constraint mechanism. Any reasoning mechanism that produces a constraint violation results in marking the context (in which the reasoning was performed) as poisoned.

(b) Blackboard Structure

The central communication medium in HEARSAY-III is the "blackboard". An application program uses the blackboard as a repository for a domain model, for representation of partial solutions, and for representation of pending activities. The blackboard is segmented into two parts:

(1) Domain blackboard

(2) Scheduling blackboard

The domain blackboard is intended as the site of competence reasoning (i.e., for reasoning within the task domain), and the scheduling blackboard is intended as the site for performance reasoning (i.e., for reasoning about scheduling). The user can further subdivide each of these blackboards.

Blackboard units are fundamental components of the representations built by application programs in HEARSAY-III. Every unit has a structure. The structures of units are used to represent unresolved decisions explicitly and such sets are called choice sets.

HEARSAY-III provides two mechanisms for resolving the ambiguity by a choice set:

(1) Deduce-mode choose.

(2) Assume-mode choose.

An application program may perform a deduce-mode choose when it has conclusive evidence that one alternative is the correct solution for the problem represented by the choice set and that there will be no desire to retract that choice based on further evidence. In this case the choice set is replaced by the alternative (their properties are merged) in the context in which the choice is made. In deduce mode, the blackboard appears as if this choice set never existed before.

An assume-mode choose also replaces the choice set with a unit that represents a merge of properties of the choice set and the chosen alternative. The difference is that an assume mode choice makes these changes in a newly created context from the one in which the choice is made. The blackboard structure in the new context is identical to that resulting from a deduce-mode choice. The choice still exists in the earlier context with its structure modified only to eliminate the alternative just chosen. In this way, if subsequent reasoning indicates that this alternative may not be best, it is possible to return to the original context and select a different alternative.

4.2.2.3 HEARSAY-III Inference Engine

The key functions of generating, combining, and evaluating hypothetical interpretations are performed by independent programs called "knowledge sources" (KSs). Each KS can be schematized as a condition-action type production rule; it reacts to blackboard changes produced by other KS executions and in turn produces new changes.

To define a KS, the user provides a triggering pattern, immediate code, and a mode. Whenever the pattern is matchable on the blackboard, HEARSAY-III creates an activation record for the KS. At the point the activation record is created, the immediate code of the KS is executed. At some subsequent time, the system's base schedule (see below) may call the HEARSAY-III Execute action on the activation record. The result of this is that the body of KS is run (executed) in the triggering context and with the pattern variables instantiated.

Each KS execution is indivisible; it runs to completion and is not interrupted for the execution of any other KS activation. This insulates the KS execution and simplifies the coding of the body; there need be no concern that during a KS execution anything on the blackboard will be modified except as effected by the KS itself.

Scheduling

Frequently, many KS activation records vie for execution and

HEARSAY-III is intended for use in domains in which KS scheduling schemes are likely to be complex and in which one might need to experiment freely with various schemes.

The scheduling blackboard at the end of each KS execution, determines which KS activation to execute next. Some KSs (known as scheduling KSs), may make changes on the scheduling blackboard to facilitate the selection of activation records. Scheduling KSs may respond to changes both on the domain blackboard and on the scheduling blackboard, including creation of activation records. The scheduling blackboard is the data base for solving the scheduling problem.

4.2.3 AGE (Attempt to Generalize)

4.2.3.1 Overview of AGE

The following discussion is a simplified version of one presented in [Hayes-Roth, et al, 83].

AGE is a software tool specifically designed to allow the implementation of a broader spectrum of KBSs. AGE gives the designer a set of separate, interconnetable, preprogrammed modules (also known as components or building blocks) for selecting a framework, implementing the knowledge base, inference engine, and the data base.

A component is a collection of LISP functions and variables that support conceptual, as well as concrete, entities. For example, the production-rule component consists of:

(1) A rule interpreter that supports syntactic and semantic description of production rule representation, and

(2) Strategies for rule selection and execution.

The components have been preprogrammed, but the designer of the KBS (or user of AGE) could modify or replace them as long as the changes conform to the definitional constraints.

The components in AGE have been carefully selected, defined, and modularly programmed to be usable in various combinations. And using different combinations makes it possible to construct programs that display different problem solving behaviors.

One particular combination (or framework) is blackboard framework. The other is backchain framework.

4.2.3.2 Blackboard Framework

A blackboard-based program written in AGE consists of three major components:

(1) The blackboard.

(2) The knowledge base.

(3) The control.

(a) Blackboard

The blackboard concept is originated from the design of HEARSAY-II, a speech understanding system [Erman, et al, 80], and

it is designed to hold input data, intermediate results and solutions. It is augmented with a variety of control and representation concepts. Some of these augmentations include production rules and object-oriented representations of knowledge, an extended blackboard, and a scheme for generating and processing expectations and goals.

(b) The Knowledge Base

The knowledge of the problem domain can be represented in two different ways:

(1) The description of the objects, both conceptual and actual.

(2) The relationships among the objects.

The knowledge to use these facts and the information on the blackboard is represented as a set of production rules. A set of related rules is called (in AGE terminology) knowledge sources (KSs).

Each production rule consists of a left-hand side (LHS) and a right-hand-side (RHS). The LHS specifies a set of conditions or patterns for the applicability of the rule. The applicability of a rule here means that either all of the specified conditions must be true, or only that some need to be true. Because of the wide range of possibilities of defining applicability, AGE asks the user to define it in the form of a function to serve as the

LHS Evaluator. An example of LHS Evaluator is all-conditions-must-be-true.

The RHS represents the implication to be drawn, under the situation specified in the LHS. These implications are represented in the form of changes to be made to the hypothesis structure (a data structure that holds input data, intermediate results, and final results), or to the knowledge base.

(c) Control Structure

In AGE several components are grouped under the heading of control. They are as follows:

(i) The Input Component: The user has to specify the format and the names of the input data, and the manner in which the data are to be acquired through this input component.

(ii) The Initialization Component: This component processes the input data and returns the name of the first KS to be invoked.

(iii) The Kernel Control Component: This component specifies the inference mechanisms to be used (discussed below).

(iv) The Termination Component: This component specifies the condition under which the program will terminate; for example, the occurrence of some specified event.

(v) The Post-Processing Component: This component is for processing after the termination of rule execution; for example, printing an hypothesis or printing an explanation.

(d) Kernel Control Component

The primary functions of kernel control are:

(1) To select an item on the blackboard to be processed next (done by inference generation subcomponent), and

(2) To invoke KSs appropriate to that item and consistent with the goal of the program (by focus of attention subcomponent).

A more detailed description of inference generation and focus of attention subcomponents can be found in [Hayes-Roth, et al, 83].

Because the control mechanisms have many details that are potentially confusing to novice user, AGE provides two rather simple, prepackaged control structures called control macros. They are useful for event driven macro and expectation driven macro control.

(e) Event-Driven Control

Event-driven control is a two step process:

(1) A rule modifies the hypothesis elements or UNITS data base and causes an event, with associated event token (which summarizes the actions to be taken by the rules).

(2) If the focused event name (assigned by the user) matches a precondition of a KS, then invoke that KS. Loop back to (1).

(f) Expectation-Driven Control

Expectation driven control is a three step process:

(1) A rule generates expectation(s).

(2) If an expectation is met, the hypothesis elements or UNITS are modified as specified. This action generates an event with an associated event token.

(3) If the focused event name matches a pre-condition of a KS, that KS is invoked. Else loop back to (1).

A more detailed description of event driven and expectation driven macros is presented in [Hayes-Roth, et al, 83].

4.2.3.3 AGE Facilities

Currently AGE is designed to be usable by persons knowledgeable in the appropriate uses of various AI problem solving methods. The user has to translate a problem into an

appropriate framework. Once a framework has been chosen, AGE provides a detailed specification of each of the components.

The AGE system consists of four major subsystems:

(a) Design Subsystem: The design subsystem guides the user in the design and construction of a application program that fits a predefined framework.

(b) KB Editor: The knowledge base editor help the user enter detailed domain specific information as well as control information for each of the components.

(c) Interpreter Subsystem: The interpreter subsystem executes the user program and provides a variety of debugging aids.

(d) Explainer Subsystem: The explainer subsystem provides a complete trace of the execution of the user program.

Chapter 5

APPLICATION CONSIDERATIONS

5.1 Introduction

Though there exists a large amount of literature about existing and developing KBS applications, the selection process for each new application requires consideration of a variety of reasons. Over the years, the knowledge engineers have developed many heuristics or intuitions. In many ways, these are similar to guidelines for building other types of software systems. They have been divided here into three major groups. First, a set of considerations that address the issues of the problem domain and the experts and users of the system that is developed for that problem domain. Next, are the technology considerations that focus on the availability of technology for implementing a KBS. Finally, are the considerations that determine whether or not the development environment and user environment are properly supportive.

The discussion is based on [Buchanan, 75], [Barnett & Bernstein, 77], and [Hayes-Roth, et al, 83].

5.2 Initial Considerations

In this section some major considerations that should be taken into account - before a decision to build KBS for a particular application is made - are discussed.

5.2.1 Task Suitability

(a) Closed Form Solution

Does the problem have a closed form solution? If a closed form solution exists and that can be implemented using other computer techniques, then KBS technology is probably not suitable. On the other hand, those other techniques may be computationally very inefficient because the number of steps involved or because of the number of possibilties (combinatorial explosion) are very high. In such a case, KBS technology can be considered.

(b) Complexity of the Problem

Is the problem too difficult or too easy? A task can be said to be "too easy", if it "takes only few minutes" and "too hard" if it requires "few months" [Hayes-Roth, et al, 83]. Though the tools and techniques to build expert systems will improve, presently it is wiser to build a system that is an expert in performing a task T in a domain D, than building a system that is an expert in domain D.

(c) Common Sense Reasoning

Does the task require considerable common sense knowledge? KBS are not general purpose problem solvers and no current system is adept at common sense reasoning. As such, it is extremely difficult to build a system that has expertise in several domains.

5.2.2 Availability of Expert

(a) Is there an expert (recognized as such in his domain) available?

One of the preconditions for building a KBS is the existence of an expert (or group of experts) in the domain being considered. If there is no expert or no one who is recognized as outstanding performer for the type of problems involved, building KBS in that domain is probably not worth considering.

(b) Can the expert be motivated to work on the development of a KBS?

The expert should be willing to give long term commitment and should become an integral part of the developing team. At the same time, he should not be expected to become an expert in computer science and KBS technology. Knowledge engineer should be willing to meet the expert at least half way.

(c) Is the knowledge engineer familiar with the problem?

The knowledge engineer should read relevent reports and talk

to other experts to learn as much about the problem domain as possible. This not only establishes a good communication between expert and knowledge engineer, but also simplifies the task of identifying the problem and expressing key concepts and relations explicitly.

5.2.3 Knowledge Acquisition Process

There are several ways of imparting domain specific knowledge to the KBS. A few them are discussed below:

(a) Interaction between knowledge engineer and expert:

The knowledge engineer should have extensive discussions with the expert in identifying the roles of participants in the knowledge acquisition process, define the problem to be attacked, and characterize goals and objectives of building a KBS. He should also watch (record) the expert's method(s) of problem solving; application(s) of formulas, heuristics, and the reduction process. This is known as a protocol study. One advantage of this approach is the ability to separate knowledge from the reasoning mechanism.

(b) Expert directly imparting knowledge into knowledge base:

With this apporach, the expert can directly interact with the KBS through a knowledge base editor, and impart knowledge directly into the KB without intervention from any one else. The assumptions are that the:

(1) Expert is familiar with the KB editor, and

(2) Expert is able to translate his expertise into the (usually) restricted syntax statements, and the expert has some knowledge about specific design specifications.

The above process may require, initially, interaction with the knowledge engineer.

TEIRESIAS is the best example for this type of approach. TEIRESIAS is a program that assists the expert to transfer his expertise to the KB. The expert carries a dialog with TEIRESIAS in a subset of natural language [Davis & Lenat, 82].

(c) Acquire knowledge directly from the data:

With this approach a separate system could be built to abstract the knowledge from the observed data and experimental results. This approach is similar to one taken in META-DENDRAL, which could infer rules about domain from the data.

The major problem with this approach is providing the necessary constraints that would limit the system to generating only rules (or knowledge) that is plausible within theory of the domain instead of all possible ones. Those rules should, of course, be consistent.

(d) Acquire knowledge directly from text books:

With this approch (described in [Badre, 73]), the knowledge acquisition mechanism should be able to read textbooks, journals,

etc. and extract the useful knowledge and transfer it into the KB. This approach may become feasible in the future.

5.2.4 Agreement With the Domain Theory

Another important factor that should be taken into consideration is whether or not there exists an underlying theory that is agreed upon by many professionals in that domain, and whether there is general agreement on what is a correct result or answer.

It is highly unlikely that a KBS will be successful if there exists many competing or even conflicting theories for a particular problem domain.

5.2.5 Expert's Model

In relation to some of the knowledge acquisition methods discussed above, one has to determine whether the expert has a model in his mind to solve the problem(s). When the expert is solving a problem, he should be able to express the steps, processes, rationale, heuristics, etc. in a reasonably orderly manner.

5.2.6 Expert's Principles of Reasoning

One has to observe whether or not the expert approaches each problem in an ad hoc manner, or applies a set of rules, heuristics and problem reduction processes that rapidly focus his attention on the key subproblems. For a KBS to be successful, it

is necessary that the expert should follow some orderly reasoning process when solving the problems.

5.2.7 Intermediate Levels of Abstraction

Many times the expert may not be expressing explicitly (or may not be even aware of) many intermediate level concepts during a problem solving activity. It is necessary that these concepts be identified. This helps organizing the KB in more efficient manner both conceptually and computationally.

5.2.8 General Vs. Domain Specific Knowledge

It is necessary to separate general knowledge from domain specific knowledge. This supports transparency and the incremental development of the system.

5.2.9 End Users

(a) Who are the end users?

None of the existing KBSs are intended for non-professionals of the domain the system was developed for. It is unlikely in the near future that systems will be developed that could be used by non-professionals and still have high performance. Therefore it is necessary for the user of the KBS to be proficient in the field, understand the underlying theory, be able to converse with the expert in the jargon of the field, and confront significant problems within the domain in his daily activities.

(b) Is a reasonable solution acceptable to the intended users?

As was mentioned few times in this report, the power of a KBS derives from its ability to reason plausibly under uncertain conditions (incomplete or inexact data) and there is no guarantee that the system will always produce a "correct" solution under those conditions. It could produce only a reasonable or plausible result. For a KBS to be successful, it is necessary that the intended users could accept such reasonable or plausible results along with their explanations.

5.2.10 Unanticipated Support

Is the domain dynamic? By this, it is meant whether the problems that users try to solve, though within the domain, are constantly shifting in unpredictable ways. Any KBS should be built with the provision for expanding its KB, but to accommodate drastic shifts may be quite difficult.

5.2.11 Cost versus Benefits

Building a KBS is expensive and time consuming. The problems that are solved by KBSs must be useful, and solutions should be reliable to the users. The time spent by the user (professional in the domain) to solve a problem using a KBS must be worth the effort.

Another aspect of usefulness of a KBS is related to data gathering and recommended actions. A KBS that can help reduce

the cost of the information gathering process and that can provide solutions with less (or low quality) input will be very useful to the users.

5.3 Technology Considerations

This section discusses some of the issues that relate to the design and implementation from a technological view point.

5.3.1 Building the Prototype System

Development of a prototype system is a very important step in construction of a KBS. The main intent of this exercise is to test whether the proposed method will work. If not, it may indicate a re-examiniation of the design or the basic underlying ideas. The prototype KB can be implemented by using whatever knowledge engineering aids are available for the chosen representation (intelligent editors, etc).

Even if the prototype system works from the beginning to the end, it does not guarantee that the final KBS will do as well across the spectrum of problems it was designed for, but it will indicate that the approach is reasonable.

5.3.2 Chunk Size

The design of the data structures and procedures should reflect as accurately as possible the expert's conceptualization of the problem domain. This not only minimizes the effort needed

for the translation, but also helps in removing errors and improving the system. This is not to suggest that the KBS should mimick (or simulate) the expert's problem solving approach; however, that the expert should be a part of the process and the system should benefit from expert's heuristic knowledge and the informal style of reasoning the expert uses.

5.3.3 Representation of Knowledge

The method for representing knowledge should be chosen carefully. Many of the successful KBSs use simple production rule representation. Inventing new representational techniques for a new application area may increase the risk of failure, unless, of course, the technique is an clear extension of a well known one. This is not to suggest that new representational techniques should not be explored, but to warn that such techniques should be rigorously tested first before they can be considered to build a large scale KBSs.

5.3.4 Inference Engine

In the beginning, at least, a simple inference engine should be built. This not only permits experimentation with the knowledge representation methods sooner, but also makes knowledge much more accessible. Some of the better known problem solving methods include heuristic search, deductive inference from rules, pattern matching.

For a very complex system with multiple levels of

abstractions and multiple representations of knowledge (like speech understanding systems, e.g., HEARSAY-II), different methods may be required to solve the problem at different levels.

5.3.5 Meta Knowledge

If the domain is very large and complex, it is increasingly difficult for anyone to stay "on top" of everything. Therefore, if the reasoning process and control can be incorporated in the inference engine, then the system will be relatively simple and easy to implement.

5.3.6 Procedural Knowledge

It is important to ensure that knowledge is not embedded in code (procedures) in the inference engine. All the knowledge should be incorporated in the system's knowledge base. This type of error in the design will reduce the flexibility of the system or force major modifications as the system grows.

5.3.7 Addition of Knowledge by the Users

If the users of a KBS add knowledge, in contrast to data (as may be necessary for solving certain problems), to knowledge base, the KBS will be difficult to design and implement - particularly the knowledge acquisition interface and associated facilities for validating the consistency of the added knowledge as well as the control mechanism in the inference engine.

5.3.8 Extensibility

A KBS should be designed to grow in various ways from its initial conception and implementation. The areas for improvement include:

(1) Increasing knowledge base.

(2) Increasing inferential capabilities.

(3) Improving the flexibility of user interface.

(4) Increasing the overall reliability and performance of the system by refining the inferential capability and learning from errors of the past.

5.3.9 Knowledge Representation Tools

(a) Generality of the tool

A tool for building a KBS should be as specialized as possible. This is because the more general the representation and control, the more difficult and inefficient is the representation of any particular chunk of knowledge.

(b) Appropriateness of the Tool

The appropriateness of a tool can be tested by building a small prototype system. Even though the actual development of the KBS may take many months of effort, it may be possible to test the effectiveness of a particular tool through the intensive efforts of the expert and knowledge engineer in a much shorter

period of time.

(c) Accessibility

A tool that is still maintaned by the developer and is proven to be robust should be selected. The selection of an old tool that is not currently mainatined by the developer may prove to be difficult to get running initially.

(d) Explanation/Interaction Facilities

If the tool selected has very good explanation and interaction facilities, it not only improves the speed of the KBS development, but also results in a more intelligible system.

(e) Problem Characteristics vs. Tool Features

The selection of a tool is directly influenced by the problem characteristics, which include size of search space, the form of data (continuous, time-varying, uncertain, inconsistent, etc.), and the structure of the problem (incomplete knowledge, interacting subproblems, etc.).

The tool selection also depends on the solution characteristics, which include the type of search (exhaustive, heuristic search, etc.), the representation of knowledge (production rules, frames, etc.), and the form of control (parallel processing of subproblems, top down refinement, etc.).

5.3.10 Design of Tools for Building KBSs

If the existing tools or aids are inadequate to build KBSs, the knowledge engineer must develop new ones. The design of such a tool involves many considerations including generality, completeness, language features, data base structure, and control methods.

(a) Generality

Generality depends on the range of application areas for which the tool is appropriate. The designers would like to develop a general purpose tool that could be used for a wide range of problems, but the tradeoff here is efficiency of design and development versus power of the tool for each application.

(b) Completeness

The completeness of the tool depends on the number and usefulness of the features included in the tool. For example, systems like EMYCIN, EXPERT, and KAS provide the largest number of special support features. These features contribute to the power and efficiency of the system within the restricted application domain.

(c) High-Level Representation Language

Providing high-level language facilities for the tool speeds up the development process and contributes to extensibility of

the system. The language should be both readable to the experts (i.e., the experts should be able to read and understand without any previous training) and manageable by the knowledge engineers (i.e., the knowledge engineer should be able to modify or augment the rules with only modest training).

(d) Explanation and Interaction Facilities

Other useful features to incorporate into tools are facilities for explanation and user interaction facilities. These facilities speed up the prototype system development.

(e) Data Representation

Another important feature of the tool is the control structure of the data base. The tool should have basic data representation schemes that is as general as possible keeping the representation task reasonably easy (constrained). If it is too restrictive, even simple problems will be unsolvable. On the other hand, if it provides too much freedom and very little guidance, complex problems will seem overly complex.

(f) Control Structure

The power, generality, and accessibility of the control mechanism are important aspects of any KBS building tool. The representation of the procedural knowledge is directly affected by the control structure. For example, the use of iteration, recursion, backward chaining, etc. affects decisions regarding

representation of procedural knowledge. A rigid and constrained control structure simplifies and speeds up the development of interaction and explanation facilities in the KBS. It also contributes to incremental development of the system, providing a higher degree of modularity than could be achieved from a more general control mechanism.

5.4 Environmental Considerations

In the last two sections, initial considerations and technology considerations were discussed. In this section, the operational and developmental environments for KBSs are discussed.

5.4.1 Interactive KBS

To be most useful to its users, a KBS is necessary that it is interactive. Even though it is possible to build a KBS that runs in a batch processing environment, it is unlikely that it will be successful; "a batch system just cannot provide helpful, rapid feedback and immediate error recovery, for example, from a simple typing error" [Buchanan, 75]. So, the basic design philosophy for a KBS should be that of a user oriented, interactive system.

5.4.2 Interactive Development Environment

An interactive development environment will speed up the

implementation process - particularly when acquiring knowledge from the expert and transferring it into knowledge base, and validating the new knowledge. Thus, it is necessary that the development environment for the KBS be an interactive one.

5.4.3 Local Operating Environment

A KBS should be able to access the local operating system and various builtin explanation and interaction facilities of the external computer environment. This fact was particularly illustrated during the development of RITA and ROSIE. Such an interaction with the external environment extends the power and generality of a KBS, since it enables the system to control other jobs in parallel, and accessing them like subroutines. For instance, this KBS can perform complex mathematical calculation in FORTRAN or access external data bases via computer networks [Hayes-Roth, et al, 83].

Chapter 6

CONCLUSIONS

The technology of KBSs has emerged from AI research. Many KBSs have been built in the past decade in a wide spectrum of application areas, from medicine and chemistry to geology and business to computer configuration and project risk assessment. The DENDRAL system has been in regular use by university and industrial chemists throughout this country. The PROSPECTOR system has been applied to many practical problems of the US Geological Survey and US Department of Energy. Digital Equipment Corporation is using the R1 system to configure their computers.

Still, KBSs have not achieved the status of being commonly known or commonly understood like many other computer-based systems.

There appears to be, as noted by Buchanan and Duda, at least three main motivations for building KBSs, apart from research purposes [Buchanan & Duda, 83]:

(a) Replication and Distribution of Expertise

An expert becomes one only after years of education, training, and experience. By building KBSs, one can provide many (electronic) copies of an expert's knowledge (or expertise), so it can be consulted even if the expert is not personally available because of geographical location, because of retirement, or for whatever reason.

(b) Union of Expertise

In some domains, there may be no single specialist whose expertise spans the entire problem domain. KBSs can provide, in one place, the union of the expertise of several specialists. For instance, PRAS (Project Risk Assessment System), being developed by Hitachi, is an expert system that can be used for planning, construction, and maintenance of large scale construction projects. It uses expertise from engineering, design, and construction specialists [Feigenbaum & McCorduck, 83].

(c) Documentation

KBSs can be used to provide a clear record of the best knowledge available for handling a specific problem and this record can be used for training.

Building KBSs is very expensive and time consuming. Construction sometimes takes as much as 10 to 25 person-years and costs as much as $1 to $2 million. But the general level of accomplishment is high enough to make it worthwhile. For instance, SRI International (with the US Geological Survey) built an expert system, PROSPECTOR, for advising during the process of field exploration for minerals. In 1982, the expert system was used by a company exploring for mining molybdenum in the Washington State Cascade Mountains, and a find was made. The value of it has been variously estimated at several million to

$100 million! [Feigenbaum & McCorduck, 83].

Still there remain a number of unresolved issues that increase the difficulties and potential risk of using KBS technology in new applications. Many of the problems and (hence) potential research areas are discussed in Chapter 7.

Chapter 7

POTENTIAL FUTURE RESEARCH AREAS

(a) Knowledge Acquisition

Knowledge acquisition (KA) is one of the most difficult and time consuming process in building KBSs. The knowledge base in DENDRAL, for instance, was originally "custom crafted" and large parts of the system were rewritten a few times as knowledge base changed. Later on, highly stylized procedures that were dependent only on global parameters were attempted. Still the programmers were required to write new procedures. Years later, finally, the knowledge of mass spectrometry was codified in production rules.

In later systems, a framework in which the vocabulary and syntax for the knowledge base are fixed is initially developed. New knowledge is filled (sometimes forced) into this framework thus speeding up the KA process considerably. The knowledge engineer is still required to interact and explain the program's framework to the expert. He is still responsible for translating the expert's problem solving knowledge into the framework. Thus, despite several concentrated efforts, the KA process still remains a bottleneck.

There have been some prototype dialogue systems with which the expert can interact and provide knowledge directly to the system without any intermediary. TEIRESIAS is one of the most successful in this respect, but even it is limited to helping debug and fill out a knowledge base that has already been largely codified.

An expert builds the knowledge base partly from past experience and textbook cases. So, it is reasonable to hope that an induction program could build a knowledge base for an expert system in a similar way. An induction program which finds meaningful, casual associations in a large data base requires considerable basic knowledge of the domain. In fact, some prototype machine learning programs already exist but none of them can be used for automatic knowledge acquisition in building KBSs. However, many prototype systems point to future research in this direction.

Ultimately, it would be desirable to have a program which can acquire knowledge directly from textbooks, journals, etc. [Badre, 73]. This process requires much more sophistication than language understanding programs posses today, including the ability to view and understand diagrams.

(b) KBS Building Tools

Though it is reasonably clear where KBS technology can be and cannot be used, there is no general theory or framework to guarantee that a selected application will be successful. How to

seclect a knowledge representation mechanism, how to select a control mechanism, how to select a KBS building tool, and if no tool exists, how to build one, are still open questions. They impact not only specific design choices, but the performance of the system as a whole.

Some KBS building tools such as AGE, HEARSAY-III, KRL, have already been developed (see Section 5.2). Though there is not enough experience with such systems to assess their value, one can expect them to play a significant role in future developments.

(c) Explanation

The success of a KBS depends, partially, on their acceptability by the users, which in turn will be influenced by the KBS's explanation facilities. The users are (typically) not computer professionals and hence cannot be expected to know the entire system. The users use a KBS as an intelligent assistant and take advise for their problems. They will make some decisions based on that advise. In many cases, they will be held responsible for their actions. Naturally, they want to know and understand the rational basis for the system's decisions [Buchanan, 82].

One kind of interactive explanation is simple question answering as described in [Scott, et al, 77]. But just answering questions about a knowledge base (known as KB static query) is not enough in giving the users the information they need. In

many complicated cases, it may be more important to know how the system uses what it knows than to know what it knows [Swartout, 77]. Thus, the user needs to be able to understand the line of reasoning (known as dynamic query of reasoning).

MYCIN is the first KBS to provide elaborate explanation facilities. But it does not take into account the differences in users level (or qualification) nor the different purposes for asking a question. Thus, it is desirable to build smarter systems that can determine and exploit those differences and provide more helpful explanations.

(d) Evaluation

In the past decade, many KBSs have been built and some of them are moving from a comfortable research and development environment into the marketplace. DENDRAL, MACSYMA, and MOLGEN all are routinely used by users who are not connected to the designers of the system. Therefore, the developers are expected to provide some objective demonstration that the system performs as well as they claim.

Existing techniques for evaluating the KBSs are few and primitive. Much more effort has been devoted to designing and constructing KBSs than to measuring their resulting performance. There is no consensus about how to evaluate KBSs (or when or why).

The criteria like correctness, efficiency, or friendliness that are used to evaluate other computer-based systems can be

used to evaluate KBSs. But they are not enough, because KBSs use human expertise and are usually compared with human performance. But this raises an important issue: whether a correct solution (for an KBS) is one that a human expert would give, one that a group of experts would agree upon, or one that represents the ideal solution (after testing and analyzing) [Hayes-Roth, et al, 83].

No one has developed a method to evaluate human expertise objectively and adequately. Though there are many kinds of tests for human experts, few of these methods seem to apply directly to the issues faced in evaluating a KBS.

It is hoped that, in the future, more attention will be directed towards the issues of evaluation.

(e) Parallel Processing

As KBSs become more complex and their knowledge bases grow in size, one needs to find methods for increasing efficiency. One way to improve efficiency is to solve subproblems in parallel. Some problems require distributed control to improve the reliability of the overall system. Very little experience exists in this direction.

(f) Learning from Experience

One way to improve the performance of a KBS is for it to learn from its past experience, the way human experts do. Any

kind of learning still requires special systems. It is desirable for every KBS to benefit from its past experience.

(g) Management of Knowledge

Maintaining a large knowledge base is as difficult as building one. In some domains where no closed form solution exists, the knowledge of an expert (along with techniques) may change. In medicine, for instance, new microbiological agents are discovered continually as well as new drugs to treat them. New techniques need to be developed to ease the maintenance of knowledge bases.

(h) Abstractions and Hierarchies

Many KBSs represent and use abstractions and hierarchies. But there is no mechanism to compare the various techniques to understand their strengths and weaknesses.

(i) Technological Innovations

With the constant innovations and improvements in computer hardware that have been taking place in the past two decades, one can expect to see "portable" expert systems, PC-based expert systems, etc. in not too distant future.

APPENDICIES

Appendix A

A CASE STUDY- MYCIN

MYCIN is medical consulting system that was developed at Stanford in 1976. A brief overview of MYCIN is presented in this appendix. The material covered here is a condensation of [Shortliffe, 76] and [Buchanan & Shortliffe, 84].

A.1 MYCIN's Problem Domain and the Users

MYCIN is a knowledge based interactive computer system to assist physicians who are not experts in prescribing antimicrobial infections of the blood (bacteremia).

An antimicrobial agent is any drug designed to kill bacteria or to arrest their growth. Thus, MYCIN assists in the selection of an agent (or combination of agents) for use in treating a patient with a bacterial infection.

The name MYCIN is taken from the common suffix shared by several of the antimicrobial agents like clindamycin, erythromycin, gentamycin, kanamycin, and vancomycin. It reflects the central concern of the program, namely the selection of an appropriate therapeutic regimen for a patient with a bacterial infection.

The problem of therapy selection and recommendation for an infectious disease is difficult and complex. First, the physician must decide whether the patient has a significant bacterial infection requiring treatment. If there is significant disease, the organism must be identified. To do this, one must obtain a specimen of the infection for culturing, analysis, and identification by a laboratory. This is a time consuming process. And, in many cases, the infection is serious enough that treatment must be begun before all of the analyses can be completed. Therefore, any recommended therapy must be based on incomplete information. To further complicate matters, the most effective drug (or a set of drugs) against the suspected or identified organism may be totally inapporpriate for the specific patient because of age or medical conditions and problems. Thus, any system or consulting physician must be aware of all of these complexities if proper advice is to be rendered in each specific case. MYCIN has been designed to cope with just such complexities and interrelationships among the many variables and to provide a physician with advise that is proper for each individual patient.

Though the problem is quite complex, the domain is well bounded. MYCIN requires knowledge related only to infectious diseases, and knowledge related to experience with various infectious organisms in terms of resistance to specific drugs, and knowledge of symptoms related to specific infections.

MYCIN is intended to be used by physicians. The dialog that

it carries on with the user is in the jargon of medicine and specifically that of infectious diseases, laboratory procedures, infectious organisms, drugs, etc. Thus, a user of MYCIN is expected to be a competent medical practitioner.

A.2 MYCIN's Knowledge Base

MYCIN's knowledge base contains several knowledge sources - production rules, clinical parameters, special functions, procedures for therapy selection and patient data base.

A.2.1 Representation of Rules

The 200 (production) rules currently in the MYCIN system consist of a PREMISE, ACTION, and sometimes an ELSE clause. Every rule has a name of the form "RULE ###", where "###" is a three digit number. The rules are stored as LISP data structures in accordance with the following Backus-Naur Form (BNF) description (only a partial description is given here; a complete description can be found in [Shortliffe, 76]):

```
<rule>      ::= <premise><action> | <premise><action><else>
<premise>   ::= ($AND<condition>...<condition>)
<condition> ::= (<func1><context><parameter>) |
                (<func2><context><parameter><value>) |
                (<special_func><arguments>) |
                ($OR<condition>...<condition>)
```

The PREMISE of a rule consists of a conjunction of conditions, each of which must hold for the indicated ACTION to be taken. Negations of conditions are handled by the individual predicates (‹func1› and ‹func2›) and therefore do not require a $NOT function to complement the Boolean function $AND and $OR. If the PREMISE of a rule is known to be false, the conclusion or action indicated by the ELSE clause is taken. If the truth of the PREMISE cannot be ascertained, or the PREMISE is false but no ELSE condition exists, the rule is simply ignored. In addition, the strength of each rule's inference is specified by certainty factor (CF) in the range -1 to +1. CF's are discussed in the next section.

A.2.2 Context Tree

Although it is common to describe a diagnosis as an inference based on attributes of the patient, MYCIN's decisions must necessarily involve not only the patient but also the cultures that have been grown, organisms isolated, and drugs that have been administered. Each of these is termed a "context" of the program's reasoning.

MYCIN currently knows about 10 different context types:

CURCULS	-	a current culture from which organisms were isolated
CURDRUGS	-	an antimicrobial agent currently being administered to a patient
CURORGS	-	an organism isolated from a current culture

OPDRGS	-	an antimicrobial agent administered to the patient during a recent operative procedure
OPERS	-	an operative procedure which the patient has undergone
PERSON	-	the patient himself
POSSTHER	-	a therapy being considered for recommendation
PRIORCULS	-	a culture obtained in the past
PRIORDRGS	-	an antimicrobial agent administered to the patient previously
PRIORORGS	-	an organism isolated from a prior culture

These context types (except for PERSON) may be instantiated more than once during any given run of the consultation program. Some may not be created at all if they do not apply to the given patient. However, each time a context tree is instantiated, it is given a unique name. For example, CULTURE-1 is the first CURCUL and ORGANISM-1 is the first CURORG. Subsequent CURCLS or PRIORCULS are called CULTURE-2, CULTURE-3, etc.

The context types instantiated during a run of the consultation program are arranged hierarchically in a data structure termed the "context tree". One such tree is shown in Figure A-1. The context types of each instantiated context is shown in parentheses besides its names. Each node in the context tree is called context and is created as an instantiation of a context type.

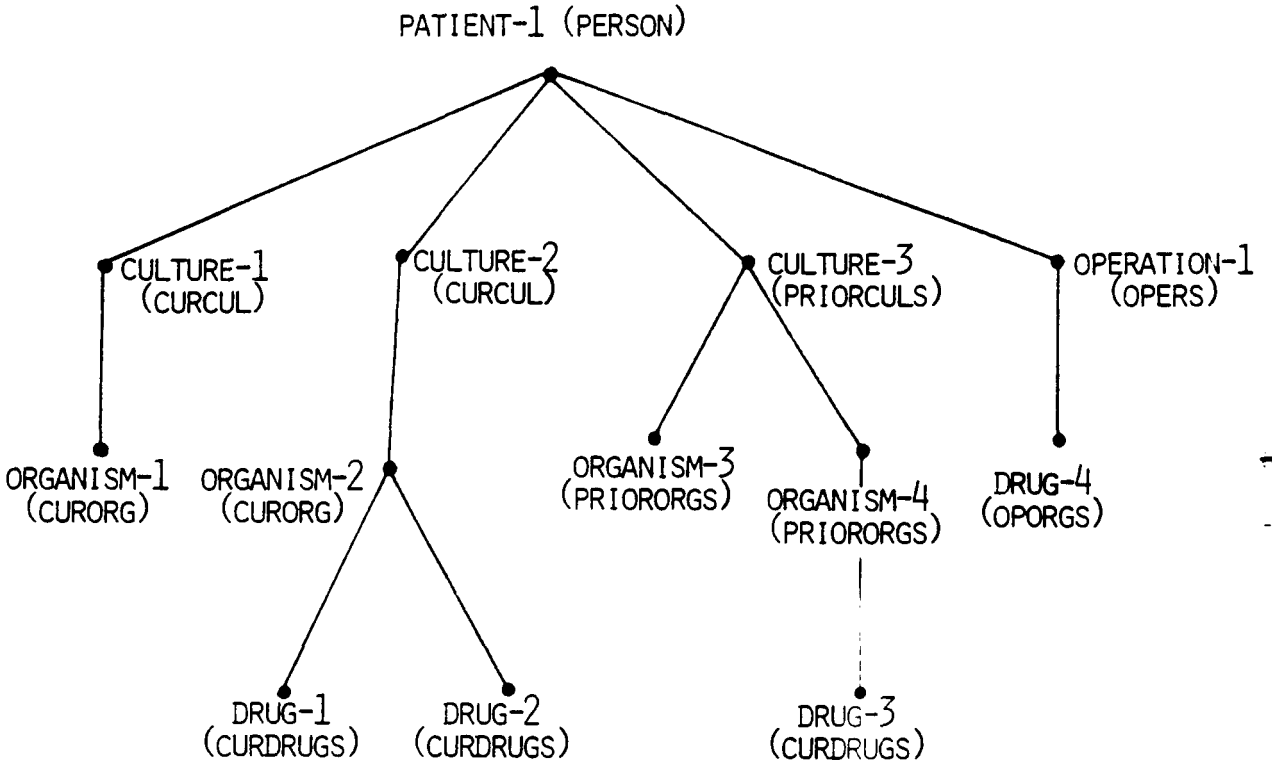


FIGURE A-1. SAMPLE CONTEXT TREE
BASED ON [BUCHANAN & SHORTLIFFE, '84]

This sample context tree corresponds to a patient from whom two current cultures and one prior culture were obtained. One organism was isolated from each of the current cultures, but the patient is being treated (with two drugs) for only one of the current organisms. Furthermore, two organisms were grown from the prior culture but therapy has included a recent operative procedure during which the patient was treated with an antimicrobial agent.

A.2.3 Categorization of Rules

The 200 rules currently used by MYCIN are not explicitly linked in a decision tree or reasoning network. This feature adheres to the designer's decision to keep the system knowledge modular and manipulable. However, rules are subject to categorization in accordance with the context - types for which they are appropriately invoked. For example, some rules deal with organisms, some with cultures, and still others deal solely with the patient himself. MYCIN's current rule categories are as follows:

(1) CULRULES - Rules that may be applied to any culture.

(2) CURCULRULES - Rules that may only be applied to current cultures.

(3) CURORGRULES - Rules that may be applied only to current organisms.

(4) DRGRULES - Rules that may be applied to any antimicrobial agent that has been administered to combat a specific organism.

(5) **OPRULES** - Rules that may be applied to operative procedures.

(6) **ORDERRULES** - Rules that are used to order the list of possible therapeutic recommendations.

(7) **ORGRULES** - Rules that may be applied to any organism.

(8) **PATRULES** - Rules that may be applied to the patient.

(9) **PDRGRULES** - Rules that may be applied to drugs given to combat prior organisms.

(10) **PRCULRUES** - Rules that may be applied only to prior cultures.

(11) **PRORGRUES** - Rules that may be applied only to isolated organisms from prior cultures.

(12) **THERULES** - Rules that store information regarding drugs of choice.

Every rule in the MYCIN system belongs to one, and only one, of these categories.

A.2.4 Clinical Parameters

The system also contains a collection of clinical parameters, represented as ‹attribute, object, value› triples. A clinical parameter is a characteristic of one of the contexts in the context tree, i.e., the name of the patient, the site of a culture, the morphology of an organism, the dose of the drug, etc. All such attributes are termed as "clinical parameters". The clinical parameters known to MYCIN are categorized in accordance with the context to which they apply. These categories include:

(1) PROP-CUL - Those clinical parameters that are attributes (e.g., site of the culture, method of collection).

(2) PROP-DRG - Those clinical parameters that are attributes of administered drugs (e.g., name of the drug, duration of administration)

(3) PROP-OP - Those clinical parameters that are attributes of operative procedures (e.g., the cavity, if any, opened during the procedure)

(4) PROP-ORG - Those clinical parameters that are attributes of organisms (e.g., identity, gram stain, morphology)

(5) PROP-PT - Those clinical parameters that are attributes of the patient (e.g., name, sex, age, allergies, diagnoses)

(6) PROP-THER- Those clinical parameters that are attributes of therapies being considered for recommendation (e.g., recommended dosage, prescribing name)

Currently there are 65 clinical parameters known to MYCIN.

Each of the parameters has a certainty factor reflecting the system's "belief" that the value is correct (an associated set of properties that is used during consideration of the parameter for a given context). This formalism is necessary because, unlike domains in which objects either have or do not have some attribute, in medical diagnosis and treatment there is often uncertainty regarding attributes such as the significance of the disease, the efficacy of a treatment or the diagnosis itself.

In addition to certainty factor, each parameter is associated with a set of properties that is used during consideration of that parameter for a given context. These properties specify such things as the:

- Range of expected values a property may have.
- The sentence to transmit to the user when requesting data from him.
- The list of rules whose PREMISEs reference the parameter.
- The list of rules whose ACTION or ELSE clauses permit a conclusion to be made regarding the parameter, etc.

Only those properties that are relevant to each parameter are associated with it. However, properly specifying how the parameter is to be represented in English is mandatory for all.

A.2.5 Simple Lists

Additional information is contained in simple lists that simplify references to variables and optimize knowledge storage by avoiding unnecessary duplication. These lists contain such things as the names of organisms known to the system and the names of normally sterile and non-sterile sites (called STERILESITES and NONSTERILESITES, respectively) from which organisms are isolated.

A.2.6 Knowledge Tables

In conjunction with a set of four special functions, MYCIN uses knowledge tables to permit a single rule to accomplish a task that would otherwise require several rules. A knowledge table contains a comprehensive record of certain clinical parameters plus the values they take on under various circumstances. For example, one of MYCIN's knowledge tables itemizes the gramstain, morphology, and aerobicity for every bacterial genus known to the system.

A.2.7 Specialized Functions

The efficient use of knowledge tables requires the existence of four specialized functions. These functions help to recommend the apparent first choice drug for the therapy.

This constitutes the majority of MYCIN's knowledge base, which permits the system to comprehend the nature of an infection without complete information about the organism involved, and provide the physician with proper advise regarding treatment under the circumstances. This organization and structure, along with the way the knowledge is used, facilitates the system's ability to explain its actions and advice.

A.3 MYCIN's Inference Engine

MYCIN's inference engine is domain independent in the sense that none of the knowledge required to provide advice about

bacteremia is embedded in it. Thus, additional rules concerning infectious disease may readily be added, or a new knowledge base could be substituted to provide therapeutic advice about a different domain of infections. As discssed in Section A.1, MYCIN's task involves a four stage decision problem:

(1) Decide which organisms, if any, are causing significant disease.

(2) Determine the likely identity of the significant organism.

(3) Decide which drugs are potentially useful.

(4) Select the best drug or drugs.

Step 1 and step 2 are closely interrelated, since determination of an organism's significance may well depend upon its presumed identity. Furthermore, MYCIN must consider the possibility that the patient has an infection with an organism not specifically mentioned by the user (for example, an occult abscess suggested by historical information or subtle physical findings). Finally, if MYCIN decides that there is no significant infection requiring antimicrobial therapy, it should skip steps 3 and 4, advising the user that no treatment is thought to be necessary.

A consultation session with MYCIN results from a simple two step procedure:

(1) Create the patient context as the top node in the context tree.

(2) Attempt to apply the goal rule to the newly created patient context.

When MYCIN first tries to evaluate the PREMISE of the goal rules, the first condition requires that it know whether there is an organism that requires therapy, MYCIN then reasons backwards in a manner that may be informally paraphrased as follows:

> How do I decide whether there is an organism requiring therapy? Well, RULE090 tells me that organisms associated with significant disease require therapy. But I don't even have any organisms in the context tree yet, so I'd better ask first if there are any organisms and if there are I'll try to apply RULE090 to each of them. However, the PREMISE of RULE090 requires that I know whether the organism is significant. I have a bunch of rules for making this decision (RULE038 RULE042 RULE044 RULE108 RULE122). For example, RULE038 tells me that if the organism came from a sterile site it is probably significant. Unfortunately I don't have any rules for inferring the site of a culture, however, so I guess I'll have to ask the user for this information when I need it...

This goal oriented approach to rule invocation and question selection is automated via two interrelated procedures, a MONITOR that analyzes rules, and a FINDOUT mechanism that searches for data needed by the MONITOR. These two procedures or components constitute MYCIN's inference engine or control structure.

MONITOR's function (Figure A-2) is to determine whether the conditions stated in the PREMISE of a rule are true. To do so, it considers each condition of the PREMISE at hand, first determining whether it has all of the necessary information to

make the determination. If it requires information, it calls FINDOUT to obtain what is needed. FINDOUT (Figure A-3) first determines whether the needed information is laboratory data. If it is, it asks the physician for it. If the physician cannot provide it, FINDOUT retrieves the list of rules that may aid in deducing the information and calls MONITOR to evaluate the rules. When the process completes, control is returned to MONITOR. If the information needed is not laboratory data, FINDOUT retrieves the list of rules that may aid in deducing the needed information and calls MONITOR to evaluate the rules. If the deductive process of applying the rules (backward from a goal to the data or information needed) cannot provide the needed information, the physician is asked to provide it. In either case, control is returned to MONITOR. Given the information that is provided by FINDOUT or that was already available, MONITOR determines whether the entire PREMISE is true. If it is not, and there is no ELSE clause, the rule is rejected. If the PREMISE is true or the ELSE clause is invoked, the conclusion stated in the ACTION of the rule or in the ELSE clause is added to the ongoing record of the consultation, and the process completes. Note that there is a recursive relationship between MONITOR and FINDOUT, such that, so long as any information is needed to evaluate a PREMISE, or rules are required to develop the needed information, the two components are in a recursively dependent and oscillating relationship until the very first rule invoked, the "goal-rule",

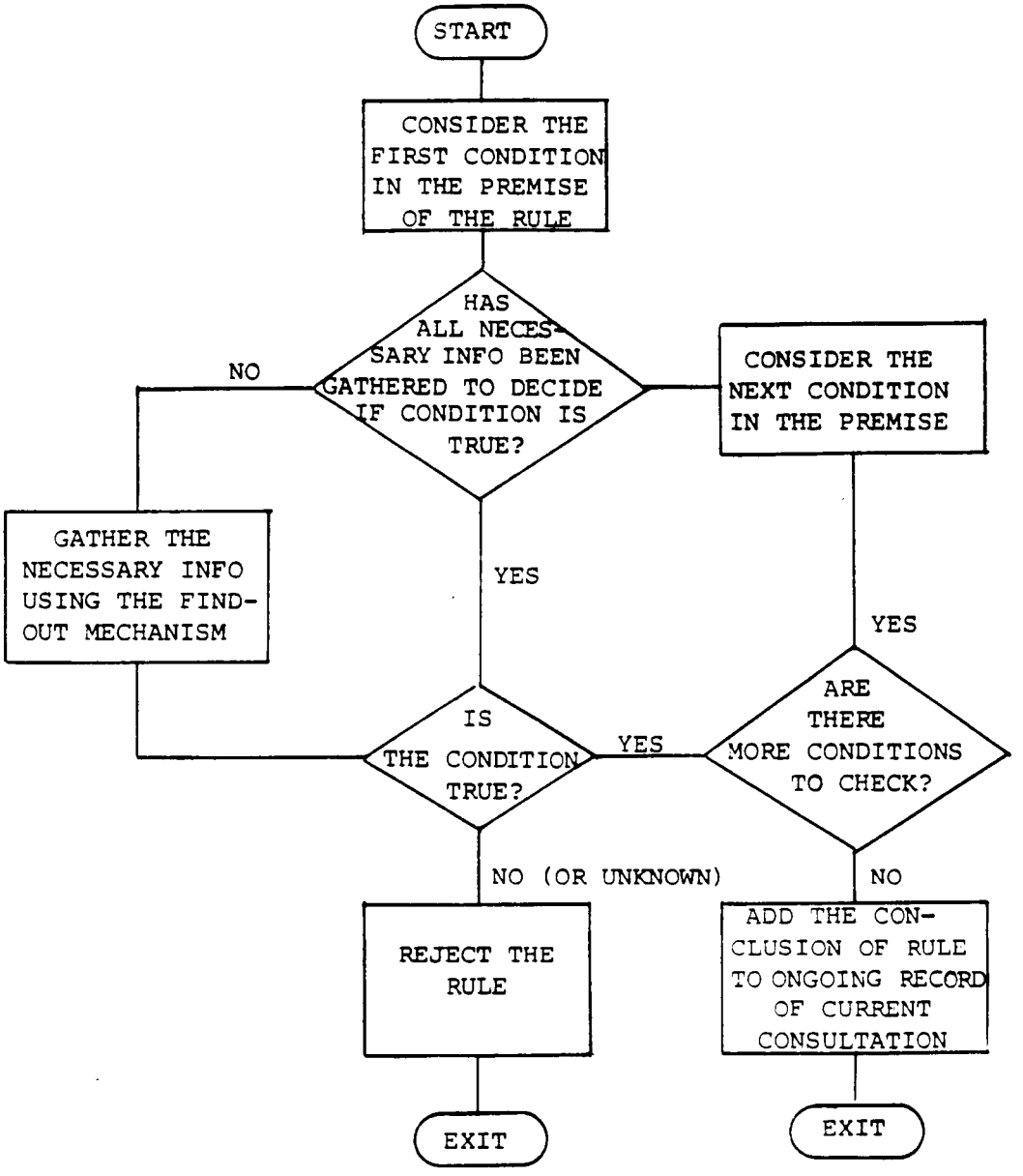


FIGURE A-2. THE MONITOR MECHANISM
BASED ON [BUCHANAN & SHORTLIFFE, '84]

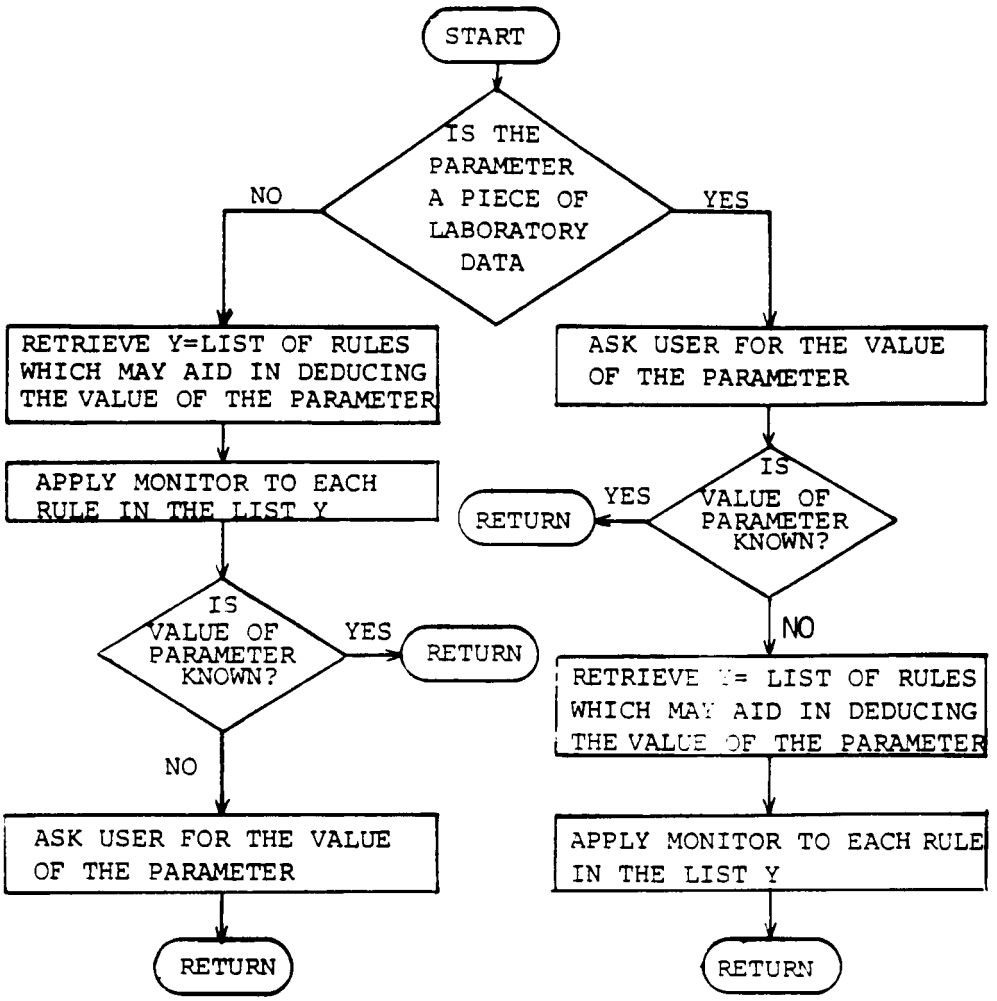


FIGURE A-3. THE FINDOUT MECHANISM
BASED ON [BUCHANAN & SHORTLIFFE, '84]

is satisfied. In the process of evaluating the rules, a great deal of related and necessary information and data are developed and retained in various tables and structures in the workspace. They serve two purposes:

(1) They prevent wasted effort that would be required to redevelop information that has already been obtained, and to prevent the system from endlessly chasing its tail.

(2) They provide the necessary history required for the explanations that may be requested by the user.

In addition to having certainty factors (CFs) for the rules and the clinical parameters in the knowledge base, the physician, when asked for either laboratory data or other information that the system itself cannot deduce, may attach a CF to his input. The default, if the physician does not provide a CF, is assumed to be +1. The certainty factors are the key to permitting MYCIN to perform inexact reasoning. The rationale, mathematics, and applications are thoroughly treated in [Shortliffe, 76]. The presentation here is very simplified.

A.4 Certainty Factors

A certainty factor (CF) is a number between -1 and +1 that reflects the degree of belief in a hypothesis. Positive CFs

indicate that there is evidence that the hypothesis is valid; the larger the CF, the greater the degree of belief. A CF = 1 indicates that the hypothesis is known to be correct. A negative CF indicates that the hypothesis is invalid; CF = -1 means that the hypothesis has been effectively disproven. A CF = 0 means either that there is no evidence regarding the hypothesis or that the evidence is equally balanced. The hypotheses in the system are statements regarding values of clinical parameters for the nodes in the context tree. To properly perform, MYCIN must deal with competing hypotheses regarding the value of its clinical parameters. To do so, it stores the list of competing values and their CFs for each node in the context tree. Positive and negative CFs are accumulated separately as measures of belief (MB) and measures of disbelief (MD) and added to form a resultant CF for a clinical parameter. The CF of a conclusion is the product of the CF of the rule that generated the conclusion and the tally of the CFs of the clinical parameters that were used in substantiating the conclusion. When a second rule supports the same conclusion, the CFs are combined by $z = x + y(1-x)$, where x is the CF of the first supporting rule, y is the CF of the succeeding rule and z is the resultant CF for the conclusion. The CFs permit the system to report findings to the physician with varying degrees of certainty such as, "There is strongly suggestive evidence that ....", "There is suggestive evidence that ....", "There is weakly suggestive evidence that ....", etc.

A.5 Context Tree

The topmost tree is always the patient. Branches are added successively to the existing nodes as FINDOUT discovers a need for them in attempting to obtain requested information for MONITOR. Thus, given only the patient, when MONITOR requests information from FINDOUT about organisms in order to evaluate the first condition in the Premise of the goal-rule, FINDOUT discovers that it cannot get organism information without having information about cultures. Thus, context(s) concerning cultures(s) are spawned from the patient node, from which eventually are spawned contexts for the organisms identified by the cultures. For those organisms deemed significant, links attach to context nodes about the relevant drugs for treating these organisms. Thus, the context tree is tailored for each patient as the system progresses through its reasoning process.

A.6 MYCIN's Explanations

One of the primary design consideration taken in MYCIN was the requirement that the system be able to explain its decisions if physicians were going to accept it. Selecting rules as the representation of the system's knowledge greatly facilitated the implementation of this capability. The physician using the system enters the explanation subsystem automatically when the consultation phase is completed, or he may enter it upon demand during the consultation session at any point at which the system

requests input from him. In the latter case, he can input "WHY" to request a detailed answer about the question just asked of him or he can input "QA" to enter the general question-answering explanation subsystem to explore the decisions and other aspects of the consultation up to the point of divergence.

The explanation provides several options to the physician. Since the system automatically enters this mode at the end of the consultation, the physician may simply input "STOP", which terminates the system. The explanation system offers several options to the user and are shown below:

Input	Question-Answering (QA) Option
HELP	Prints this list.
EQ	Explain a specific question asked of the physician during the consultation - each has a sequence number, which must accompany the EQ request.
IQ	Is a prefix for a question about information acquired by the system during the consultation. The question is phrased in the limited English that MYCIN can handle.
NOPREFIX	A general question is assumed being asked about the content of MYCIN's rules.
PR	Requests a particular rule be printed and must be followed by the rule number.
STOP	Exit from explanation system.
RA	Permits entry to the rule acquisition module for recognized experts.

An Example: Suppose a physician wants explanation for question 48. Then he inputs "EQ 48". To which the system would respond:

QUESTION 48 WAS ASKED IN ORDER TO FIND OUT THE PATIENT'S DEGREE OF SICKNESS (ON A SCALE OF 4) IN AN EFFORT TO EXECUTE RULE068. He may then optionally input "PR68" or "WHAT IS RULE068" to see what exactly was being sought and why.

A.7 MYCIN's Interfaces

MYCIN has two interfaces. One is for the using physician, through which he may answer questions posed by the system and ask questions of it; the other is a knowledge-acquisition interface accessible only to experts recognized as such by the system.

All of the questions asked of the user have been carefully designed not to require the language-understanding component. Thus, instead of asking, "What is the infectious disease diagnosis for the patient?" it asks, "Is there evidence that the patient has a meningitis?" To which only a simple "yes" or "no" is required.

The knowledge-acquisition interface, on the other hand, permits the expert to input a new rule in stylized English, with prompting to obtain the rule in the proper sequence: Premise first, condition by condition, followed by the Action, and then an Else clause if one is required. The system then translates the rule into internal form, reordering the conditions of the Premise if necessary, according to a set of criteria developed to improve the rule-evaluation process. It then retranslates the rule into English and requests that the expert decide whether the

rewritten version was the one intended. If not, the expert may modify selected parts and is not required to restate the entire rule unless there has been a gross misunderstanding.

The same mechanism is used when an expert wants to correct or modify an existing rule. In all cases, when a new or corrected rule has been approved by the expert, the system checks to see whether the rule is consistent with the existing rule set. If the new or modified rule subsumes or is subsumed by an existing rule, it is not readily discoverable, and no test is made for this condition. If a rule is discovered to be in conflict with an existing rule, it is rejected.

A.8 Evaluation of MYCIN

MYCIN's performance has been externally evaluated. There have been different empirical studies of MYCIN's performance, each simpler than the previous but all of them time consuming. The last one was reported in [Yu, et al, 79]. The following discussion is based on [Yu, et al, 79] and [Buchanan, 82].

Ten meningitis cases were selected randomly and their descriptions were presented to seven Stanford physicians and one student. They were asked to give their therapy recommendations for each case. Those recommendations along with MYCIN's recommendations for each case and actual therapy were collected in 10 x 10 matrix - ten cases each with ten recommendations. The a panel of experts not at Stanford, were asked to give each

recommendation a zero if, in his opinion, it was unacceptable for the case and one if the recommendation was acceptable. They did not know, which, if any, recommendation came from a computer. The results are shown in the Table A-1.

Table A-1. Ratings of Antimicrobial Selection by 8 Experts on 10 Meningitis Cases* [Buchanan & Shortliffe, 84]

Prescribers	No (%) Of Items In Which Therapy Was Rated Acceptable By An Evaluator
MYCIN	52 (65)
Faculty-1	50 (62.5)
Faculty-2	48 (60)
Infectious Disease Fellow	48 (60)
Faculty-3	46 (57.5)
Actual Therapy	46 (57.5)
Faculty-4	44 (55)
Resident	36 (45)
Faculty-5	34 (42.5)
Student	24 (30.5)

* Perfect Score = 80; Unacceptable Therapy = 0; Equivalent or Acceptable Alternate = 1.

As can be seen from the table, the difference between MYCIN's score and the score of the infectious disease experts at Stanford is not significant. Thus, the designers of MYCIN claim to have shown that MYCIN's recommendations were viewed by outside experts to be as good as the recommendations of the local experts, and all of those better than the recommendations of physicians (and the student) who are not meningitis experts.

Additional useful reference related to MYCIN are: [Shortliffe, 76], [Yu, et al, 79], [Buchanan, 82], and [Buchanan & Shortliffe, 84].

Appendix B

LIST OF EXPERT SYSTEMS

The following list of expert systems is based on [Michie, 84].

NAME OF SYSTEM OR PROJECT	APPLICATION AREA	BRIEF DESCRIPTION	REFERENCES
AGE	Knowledge Engineering	Provides guidance on building expert systems and a set of tools for doing so.	[Nii & Aiello, 79]
AM	Knowledge Engineering	Generates new mathematical formulas, terms, etc.	[Davis & Lenat, 82]
AL/X	Knowledge Engineering	A domain-independent development of MYCIN and PROSPECTOR usable for developing rule-based consultation programs for many fields.	[Reiter, 81]
CASNET	Medicine	Long-term management of glaucoma.	[Weiss, 81]
CENTAUR	Medicine	Interprets pulmonary function test measurements from patients with lung disorders.	[Aikins, 80]
CRIB	Fault Diagnosis	Diagnosis of faults in computer hardware and software.	[Addis, 80]
CRYSALIS	Science	Infers the structure of a protein from a map of electron density derived from x-ray crystallographic data	[Feigenbaum & Engelmore, 77]
DART	Engineering	Diagnosing hardware faults in computer systems.	Under development at Stanford

NAME OF SYSTEM OR PROJECT	APPLICATION AREA	BRIEF DESCRIPTION	REFERENCES
DENDRAL	Science	Identification of organic compounds by analysis of mass spectrogram.	[Feigenbaum, et al, 71]
EMYCIN	Knowledge Engineering	A domain-independent version of MYCIN, Usable for developing rule-based consultation programs for many fields.	[Van Melle, et al, 81]
EXPERT	Knowledge Engineering	A system for designing and building models for consultation.	[Weiss & Kulikowski, 79]
EXSEL	Computing	Configuring the VAX/780 computer system.	[McDermott, 82]
GA1	Science	Infers DNA structures from pieces (segments) of structures.	[Stefik, 78]
GAMMA	Science	Interpreting gamma ray activation spectra.	[Barstow, 79]
GUIDON	Knowledge Engineering (Education)	Case-method tutor designed to improve a student's ability to diagnose complex problems in medicine and science.	[Clancey, 82]
HEADMED	Medicine	Psychopharmacology advisor (constructed using MYCIN).	[Heiser, et al, 78]
INTERNIST	Medicine	Diagnosis in internal medicine.	[Pople, 77]
MACSYMA Advisor	Mathematics	An automated consultant for MACSYMA (an algebraic manipulation system).	[Genesereth, 78] [Moses, 75]
MDX	Medicine	Performs diagnoses related to cholestasis.	[Chandra-sekaran, 79]

NAME OF SYSTEM OR PROJECT	APPLICATION AREA	BRIEF DESCRIPTION	REFERENCES
META-DENDRAL	Science	Induces rules for determining molecular structure from mass spectrometry data.	[Buchanan & Feigenbaum, 78]
MOLGEN	Science	Provides intelligent advise to a molecular geneticist on the planning of experiments involving the manipulation of DNA.	[Martin, et al, 77]
MYCIN	Medicine	Diagnoses certain infectious diseases and recommends appropriate drug treatment.	[Shortliffe, 76]
ONCOCIN	Medicine	Assists in the management of cancer patients on chemotherapy protocols for forms of lymphoma.	[Shortliffe, et al, 81]
PROS-PECTOR	Geology	Aids geologists in evaluating mineral sites for potential deposits.	[Hart & Duda, 78]
PSYCO	Knowledge Engineering (Medicine)	Experimental production system compiler.	[Fox & Rector, 82]
PUFF	Medicine	Analyses results of pulmonary function tests for evidence of possible pulmonary function disorder.	[Kunz, et al, 78]
R1	Knowledge Engineering	A domain independent system for production	[McDermott, 80]
RAFFLES	Fault Diagnosis	Diagnosis of faults in computer hardware and software.	[Addis, 80]

NAME OF SYSTEM OR PROJECT	APPLICATION AREA	BRIEF DESCRIPTION	REFERENCES
RITA	Knowledge Engineering	Provides the user with a language for defining intelligent interfaces to external data systems.	[Anderson & Gillogly, 76]
RLL	Knowledge Engineering	Provides the user with a flexible set of facilities as a tool for building his own knowledge representation language.	[Greiner & Lenat, 80]
SACON	Engineering	Advises structural engineers in using the structural analysis program MARC.	[Bennett & Engelmore, 79]
SECS	Science	Proposes schemes for synthesizing stated organic compounds.	[Wipke, et al, 77]
SU/X	Engineering	Forms and updates hypotheses about location, velocity, etc. of objects from primary signal data (spectra).	[Nii & Feigenbaum, 78] [Nii, et al, 82]
TEIRESIAS	Medicine	Knowledge acquisition program used with MYCIN.	[Davis & Lenat, 82]
UNITS	Knowledge Engineering	Interactive language providing general-purpose facilities for knowledge representation. Used for MOLGEN plus other small applications.	[Stefik, 80]
VLSI	Engineering	Assistance in the design of very large scale integrated circuits.	Under development at Stanford
VM	Medicine	Provides diagnostic and therapeutic suggestions for critical care of patients needing mechanical assistance with breathing.	[Fagan, 80]

Appendix C

FIFTH GENERATION PROJECT

As was mentioned in the beginning of this thesis, in the past decade, there had been a major shift in AI research. It was from a search for broad, general laws of thinking toward an appreciation of specific knowledge - facts, experiential knowledge, and how to use knowledge - as the central issue in intelligent behavior. In addition to this shift, in recent years, there has been a great deal of discussion on the growing need for a new generation of computers. In 1981, a research project known as "Fifth Generation Computer Systems" was started in Japan to further the research and development of the next generation of computers. The Japanese believe that the computers of the next decade will be used increasingly for non-numeric data processing such as symbol manipulation and applied AI (KBSs) [Moto-oka & Stone, 84]. This appendix provides a brief introduction to the Fifth Generation Project, its organization, its funding, various phases of the project, and its major goals. The presentation in this appendix is based on the book "The Fifth Generation" by Edward Feigenbaum and Pamela McCorduck [Feigenbaum & McCorduck, 83], and on [McCorduck, 83].

In October 1981, Japan's Ministry of International Trade and Industry (MITI) sponsored a conference to announce a new national project. Alongside national projects in supercomputing and robotics, there would be an effort to develop a new generation (the fifth, by their reckoning) of computers.

The Fifth Generation is a consortium of eight firms (Fujitsu, Hitachi, Nippon Electric Corporation, Mitsubishi, Matsushita, Oki, Sharp, and Toshiba) and two national laboratories (the government-owned Nippon Telephone and Telegraph's Musashino Laboratories, and MITI's own Electrotechnical Laboratory). Approximately forty hand-picked researchers from each of the firms and laboratories gathered under one roof in Tokyo in April 1982 at the new Institute for New Generation Computer Technology (ICOT). Their director is Kazuhiro Fuchi, who came from the Electrotechnical Laboratory and was the intellectual spirit behind the Fifth Generation Project.

At the present all funds come from MITI. Although a national project is normally a partnership of government and private funds, the firms participating the Fifth Generation Project argued that they could not afford to support such a high-risk project and supply top researchers too. MITI agreed, and is underwriting the project for the first three years. ICOT's second-year budget is $13.6 million, up significantly over the first year's budget of $2 million. Across the ten-year period of the project, assuming typical contributions from the firms, the total budget will probably approach $200 million.

The fifth generation of computers will not be traditional computers. Instead, they will be symbolic inference machines, capable of reasoning their way swiftly through massive amounts of knowledge and data. They will be computers that can learn, associate, make inferences, make decisions, and otherwise behave in ways usually considered the exclusive province of human reason. Even their name signals the change: knowledge information processing systems, or KIPS. KIPS will be the engines of the information society; small, robust and inexpensive. They will appear as universal appliances, as commonplace and easy to use as the telephone.

The project's ten-year plan is divided into three successive stages. The first three-year stage is devoted to the development of a prototype machine, a personal PROLOG workstation that will have a knowledge base comparable to present-day expert systems (thousands of rules and thousands of objects) but whose reasoning powers will be a million logical inferences per second (LIPS), an order of magnitude improvement over software-based PROLOG implementations on today's common mainframe computers such as the DEC 2060. The prototype should be finished sometime this year, with commercial products due a year or so later. This first phase is Japan's opportunity to climb the learning curve, and is explicitly planned for that purpose.

The second four-year stage is for engineering experimentation, prototyping, continuing experiments at significant applications, and the initial experiments at systems

integration. The first thrust at the major problems of parallel processing will be done in those years.

The final three-year phase will concentrate on advanced engineering, building the final major engineering prototypes, and further systems integration work. The ultimate goal, scheduled for the early 1990s, is nothing less than an inference supercomputer, capable of a million to a billion LIPS, with a knowledge base that can handle tens of thousands of inference rules and hundreds of millions of objects - about the right size to encompass the Encyclopedia Britannica. The Japanese will rely heavily on bootstrapping: the project's earlier work on CAD will be used in later hardware design, for example.

Fifth Generation machines will understand spoken, written, and graphical input. The Japanese are launching intensive research and development into intelligent interfaces, including natural language processing, speech understanding, and graphics and image understanding.

Speech understanding research, for example, will cover speech wave analysis, semantic analysis, and pragmatic analysis (which derives understanding by extracting themes in a given sentence by detecting focus shifts, and so on). Eventually the machine will be expected to understand continuous human speech with a vocabulary of 50,000 words and 95 percent accuracy from a few hundred or more speakers. The speech understanding system is also expected to be capable of running a voice activated typewriter, and of conducting a dialogue with users by means of

synthesized speech in Japanese or English.

Text analysis is also considered part of natural language processing by the Japanese, although they are aware that the techniques used for large-scale text analysis are different from the techniques needed to smooth the way for an individual user to talk to a machine. This work also involves a highly ambitious machine translation program (initially between English and Japanese) with a vocabulary of 100,000 words. The goal is 90 percent accuracy (the remaining ten percent to be processed by humans). Translations will be the product of an integrated system that takes part in each of the processes from the compilation of the text to printing the translated documents.

Picture and image processing are considered almost as important as language processing, especially as they contribute to CAD/CAM and the effective analysis of aerial and satellite images, medical images, and the like. Eventually the image understanding system is expected to store about 100,000 images. In this, as in voice recognition, the Japanese are building on superb R&D that they did themselves in the 1970s during the Pattern Information Processing Systems (PIPS) national project.

The Fifth Generation Project has captured the imagination of computer scientists around the world (almost all major computer journals carried "special issues" on the Fifth Generation Project), and even began to attract popular attention (major articles have recently appeared in NEWSWEEK, TIME, BUSINESSWEEK, FORTUNE).

At the heart of the Fifth Generation Project are KBSs. This thesis addressed major issues, concepts, and techniques related to KBSs. As was discussed in Chapter 7, numerous problems exist in building, maintaining, and modifying large-scale KBSs. In addition to these, the Fifth Generation Project faces major challenges in parallel architectures, distributed functions, VLSI design and fabrication.

REFERENCES

[Addis, 80]. T. Addis, "Towards an Expert Diagnostic System", ICL Technical Journal, vol. 2, 1980, pp. 79-105.

[Aikins, 80]. "Prototypes and Production Rules: A Knowledge Representation for Computer Consultations", Ph.D. Dissertation, Report No. STAN-CS-80-814, Computer Science Department, Stanford University, Stanford, CA, 1980.

[Anderson & Gillogly, 76]. R. Anderson and J. Gillogly, "The RAND Intelligent Terminal (RITA) as a Network Access Aid", Proc. American Federation of Information Processing Society (AFIPS), vol. 45, 1976, pp. 501-509.

[Badre, 73]. N. Badre, "CLET: A Computer Program That Learns Arithmetic From An Elementary Textbook", IBM Research Report, IRC 4235, 1973.

[Barnett, 75]. J. Barnett, "A Phonological Rules System", Technical Memo, TM-5478/000/00, System Development Corporation, Santa Monica, CA, 1975.

[Barnett & Bernstein, 77]. J. Barnett and M. Bernstein, "Knowledge Based Systems: A Tutorial", Technical Report, TM-(L)-5903/000/000, System Development Corp., Santa Monica, CA, 1977.

[Barnett, et al, 80]. J. Barnett, M. Bernstein, R. Gillman, and I. Kameny, "The SDC (System Development Corporation) Speech Understanding System", in Trends in Speech Recognition, W. Lea (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1980, pp. 272-293.

[Barr & Feigenbaum, 81]. A. Barr and E. Feigenbaum, (Eds.), The Handbook of Artificial Intelligence (vols. I and II), Kaufman, Los Altos, CA, 1981.

[Barstow, 79]. D. Barstow, "Knowledge Engineering in Nuclear Physics", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 34-36.

[Bennet & Engelmore, 79]. J. Bennett, and R. Engelmore, "SACON: A Knowledge-Based Consultant for Structural Analysis", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 79-81.

[Bobrow & Winograd, 77]. D. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language", Cognitive Science, vol. 1, no. 1, 1977.

[Brachman & Smith, 80]. R. Brachman and B. Smith, "Special Issue on Knowledge Representation", SIGART Newsletter, no. 90, Feb. 1980.

[Brachman, 83]. R. Brachman, "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks", Computer, vol. 16, no. 10, 1983, pp. 30-36.

[Brown, et al, 83]. J. Brown, R. Burton, and J. de Kleer, "Knowledge Engineering and Pedagogical Techniques in SOPHIE (Sophisticated Instructional Environment) I, II, and III", in Intelligent Tutoring Systems, D. Sleeman, and J. Brown (Eds.), Academic Press, London, 1983.

[Buchanan, 75]. B. Buchanan, "Applications of Artificial Intelligence to Scientific Reasoning", Proc. Second USA Japan Computer Conference, Tokyo, Japan, 1975.

[Buchanan & Feigenbaum, 78]. B. Buchanan and E. Feigenbaum, "DENDRAL and META-DENDRAL: Their Applications Dimension", Artificial Intelligence, vol. 11, 1978, pp. 5-24.

[Buchanan & Barstow, 81]. B. Buchanan and D. Barstow, "Maxims for Knowledge Engineering", Report No. HPP 81-4, Computer Science Department, Stanford University, Stanford, CA, 1981.

[Buchanan, 82]. B. Buchanan, "New Research on Expert Systems", in Machine Intelligence (vol 10), J. Hayes, D. Michie, and Y. Pao (Eds.), Ellis Horwood, Chichester, England, 1982, pp. 269-299.

[Buchanan & Duda, 83]. B. Buchanan and R. Duda, "Principles of Rule-Based Expert Systems", in Advaces in Computers (vol. 22), M. Yovits (Ed.), Academic Press, New York, NY, 1983.

[Buchanan & Shortliffe, 84]. B. Buchanan and E. Shortliffe (Eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, Reading, MA, 1984.

[Chairniak, et al, 79]. E. Chairniak, C. Riesbeck, and D. McDermott, Artificial Intelligence Programming, Erlbaum, Hillsdale, NJ, 1979.

[Chandrasekaran, 79]. B. Chandrasekaran, "An Approach to Medical Diagnosis Based on Conceptual Structures", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979.

[Chilausky, et al, 76]. R. Chilausky, B. Jacobsen, and R. Michalski, "An Application of Variable-Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules", Proc. Sixth Annual Int'l. Symp. Multiple-Valued Logic, 1976.

[Clancey, 82]. W. Clancey, "Tutoring Rules For Guiding A Case Method Dialogue", in Intelligent Tutoring Systems, D. Sleeman and J. Brown (Eds.), Academic Press, London, 1982.

[Cohen & Feigenbaum, 82]. P. Cohen and E. Feigenbaum, (Eds.), The Handbook of Artificial Intelligence (vol. III), Kaufman, Los Altos, CA, 1982.

[Davis, et al, 75]. R. Davis, B. Buchanan, and E. Shortliffe, "Production Rules as Representation for a Knowledge-Based Consultation Program", Stanford AI Laboratory Memo, AIM-266, Report No. STAN-CS-75-519, Computer Science Department, Stanford University, Stanford, CA, 1975.

[Davis, 76]. R. Davis, "Application of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases", Ph.D. Dissertation, Stanford AI Laboratory Memo, AIM-283, Computer Science Department, Stanford University, Stanford, CA, 1976.

[Davis, et al, 77]. R. Davis, B. Buchanan, and E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program", Artificial Intelligence, vol. 8, no. 1, 1977, pp. 15-45.

[Davis & King, 77]. R. Davis and J. King, "An Overview of Production Systems", in Machine Intelligence (vol. 8), E. Elock and D. Michie (Eds.), Horwood, Chichester, England, 1977, pp. 300-332.

[Davis, 81]. R. Davis, "The Dipmeter Advisor: Interpretation of Geological Signals", Proc. Seventh Int'l. Joint Conf. on Artificial Intelligence (IJCAI-7), 1981.

[Davis & Lenat, 82]. R. Davis and D. Lenat, Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, NY, 1982.

[Duda & Gashing, 81]. R. Duda and J. Gashing, "Knowledge-Based Expert Systems Come of Age", BYTE, vol. 6, September 1981, pp. 238-281.

[Englemore & Nii, 77]. R. Englemore and H. Nii, "A Knowledge-Based System for the Interpretation of Protein X-ray Crystallographic Data", Report No. STAN-CS-77-589, Computer Science Department, Stanford University, Stanford, CA, 1977.

[Englemore & Terry, 79]. R. Englemore and A. Terry, "Structure and Function of the CRYSALIS System", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 250-256.

[Erman, et al, 80]. L. Erman, F. Hayes-Roth, V. Lesser, and R. Reddy, "The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", Computing Surveys, vol. 12, no. 2, 1980, pp. 213-253.

[Erman, et al, 81]. L. Erman, P. London, and S. Fickas, "The Design and an Example Use of HEARSAY-III", Proc. Seventh Int'l Joint Conf. on Artificial Intelligence (IJCAI-7), 1981, pp. 409-415.

[Evans, 64]. A. Evans, "An ALGOL 60 Compiler", in Annual Review of Automatic Programming, R. Goodman (Ed.), vol. 4, 1964, pp. 87-124.

[Fagan, et al, 79]. L. Fagan, J. Kunz, E. Feigenbaum, and J. Osborn, "Representation of Dynamic Clinical Knowledge: Measurement Interpretation in the Intensive Care Unit", Proc. Sixth Int'l Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 260-262.

[Fagan, 80]. L. Fagan, "VM: Representing Time-Dependent Relations in a Clinical Setting", Ph.D. Dissertation, Computer Science Department, Stanford University, Stanford, CA, 1980.

[Fain, et al, 81]. J. Fain, D. Gorlin, F. Hayes-Roth, S. Rosenschein, H. Sowizral, and D. Waterman, "The ROSIE Language Reference Manual", Technical Report No. N-1647-ARPA, RAND Corporation, Santa Monica, CA, 1981.

[Feigenbaum, 63]. E. Feigenbaum, "Simulation of Verbal Learning Behavior", in Computers and Thought, E. Feigenbaum and R. Feldman (Eds.), McGraw-Hill, San Francisco, CA, 1963, pp. 297-309.

[Feigenbaum, et al, 71]. E. Feigenbaum, B. Buchanan, and J. Lederberg, "On Generality and Problem Solving: A Case Study Using the DENDRAL Program", in Machine Intelligence (vol 6), B. Meltzer and D. Michie (Eds.), Edinburgh University Press, Edinburgh, 1971, pp. 165-190.

[Feigenbaum, et al, 77]. E. Feigenbaum, R. Englemore, and C. Johnson, "A Correlation Between Crystallographic Computing and Artificial Intelligence Research", Acta Crystallographica, vol. A33, no. 13, 1977.

[Feigenbaum & McCorduck, 83]. E. Feigenbaum and P. McCorduck, The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World, Addison-Wesley, Reading, MA, 1983.

[Feldman, et al, 72]. J. Feldman, J. Low, D. Swinehart, and R. Taylor, "Recent Developments in SAIL", Report No. STAN-CS-308, Computer Science Department, and Report No. AIM-176, AI Laboratory, Stanford University, Stanford, CA, 1972.

[Fikes, et al, 72]. R. Fikes, P. Hart, and N. Nilsson, "Learning and Executing Generalized Robot Plans", Artificial Intelligence, vol. 3, 1972, pp. 251-288.

[Filman & Weyhrauch, 76]. R. Filman and R. Weyhrauch, "An FOL Primer", Stanford AI Laboratory Memo AIM-288, AI Laboratory, Stanford University, Stanford, CA, 1976.

[Floyd, 61]. R. Floyd, "A Descriptive Language for Symbol Manipulation", Journal of ACM, vol. 8, 1961, pp. 579-584.

[Forgy, 76]. C. Forgy, "A Production System Monitor for Parallel Computers", Computer Science Department, Technical Report, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[Forgy & McDermott, 77]. C. Forgy and J. McDermott, "OPS, a Domain-Independent Production System Language", Proc. Fifth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-5), 1977, pp 933-939.

[Forgy, 80]. C. Forgy, "The OPS5 User's Manual", Computer Science Department, Technical Report, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[Fox & Rector, 82]. J. Fox and A. Rector, "Expert Systems for Primary Medical Care?", Automedica, vol. 4, no. 2 & 3, 1982, pp. 123-130.

[Genesereth, 78]. M. Genesereth, "Automated Consultation for Complex Computer Systems", Ph.D. Dissertation, Harvard University, Cambridge, MA, 1975.

[Gevarter, 83]. W. Gevarter, "Expert Systems: Limited but Powerful", Spectrum, 1983.

[Goldberg & Weiss, 80]. R. Goldberg and S. Weiss, "An Experimental Transformation of a Large Expert System Knowledge-Base", Working Paper, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1980.

[Goldman, 78]. N. Goldman, "AP3 User's Guide", Information Sciences Institute, University of Southern California, Los Angeles, CA, 1978.

[Green, 69]. C. Green, "The Application of Theorem-Proving to Question-Answering Systems", Proc. First Int'l. Joint Conf. on Artificial Intelligence (IJCAI-1), 1969, pp. 219-237.

[Greiner & Lenat, 80]. R. Greiner and D. Lenat, "A Representation Language Language", Proc. American Association of Artificial Intelligence, (AAAI), vol. 1, 1980, pp. 165-169.

[Hanson & Riseman, 78]. A. Hanson and E. Riseman, "VISIONS: A Computer System for Interpreting Scenes", in Computer Vision Systems, A. Hanson and E. Riseman (Eds.), Academic Press, New York, NY, 1978.

[Hart, et al, 68]. P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on System Science Cybernetics, vol. 4, no. 2, 1968, pp. 100-107.

[Hart, et al, 78]. P. Hart, R. Duda, and M. Einaudi, "A Computer Based Consultation System for Mineral Exploration", Technical Report, SRI International, Menlo Park, CA, 1978.

[Hayes-Roth, et al, 83]. F. Hayes-Roth, D. Waterman, and D. Lenat, (Eds.) Building Expert Systems, Addison-Wesley, Reading, MA, 1983.

[Heiser, et al, 78]. J. Heiser, R. Brooks, and J. Ballard, "Progress Report: A Computerized Psychopharmacology Advisor", Proc. 11th Collegium Internationale Neuro-Psychopharmacologicum, Vienna, Austria, 1978.

[Hendrix, 77]. G. Hendrix, "LIFER: A Natural Language Interface Facility", Proc. Fifth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-5), 1977, pp. 183-191.

[Hewitt, 71]. C. Hewitt, "Description and Theoretical Analysis (Using Schemas) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot", Ph.D. Dissertation, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1971.

[Hewitt, 84]. C. Hewitt, "Design Issues in Parallel Architectures for Artificial Intelligence", Proc. Seventeenth Hawaii Int'l. Conf. on System Sciences (HICSS-17), vol. 1, 1984, pp. 418-423.

[Klahr, 78]. P. Klahr, "Planning Techniques for Rule Acquisition in Deductive Question Answerings", in Pattern Directed Inference Systems, D. Waterman and F. Hayes-Roth, (Eds.), Academic Press, New York, NY, 1978.

[Kuipers, 75]. B. Kuipers, "A Frame for Frames: Representing Knowledge for Recognition", in Representation and Understanding: Studies in Cognitive Science, D. Bobrow and A. Collins (Eds.), Academic Press, New York, NY, 1975.

[Kunz, et al, 78]. J. Kunz, R. Fallat, D. McClung, J. Osborn, B. Votteri, H. Nii, J. Aikins, L. Fagan, and E. Feigenbaum, "A Physiological Rule-Based System for Interpreting Pulmonary Function Test Results", Report No. HPP-78-19, Computer Science Department, Stanford University, Stanford, CA, 1978.

[Larkin, et al, 80]. J. Larkin, J. McDermott, D. Simon, and A. Simon, "Expert and Novice Performance in Solving Physics Problems", Science, vol. 208, no. 6, 1980, pp. 1335-1342.

[Le Faivre, 77]. R. Le Faivre, "FUZZY Reference Manual", Computer Science Department, Rutgers University, New Brunswick, NJ, 1977.

[Lowerre & Reddy, 80]. B. Lowerre and R. Reddy, "The HARPY Speech Understanding System", in Trends in Speech Recognition, W. Lea (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1980, pp. 340-360.

[Malhotra, 75]. A. Malhotra, "Knowledge-Based English Language System for Management Support: Analysis of Requirements", Proc. Fourth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-4), 1975, pp. 842-847.

[Martin, et al, 77]. N. Martin, P. Friedland, J. King, and M. Stefik, "Knowledge-Base Management for Experiment Planning in Molecular Genetics", Proc. Fifth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-5), 1977, pp. 882-887.

[McCalla & Cercone, 83]. G. McCalla and N. Cercone, "Approaches to Knowledge Representation", Computer, vol. 16, no. 10, 1983.

[McDermott, 74]. D. McDermott, "Assimilation of New Information", MIT AI Laboratory Report No. AI-TR-291, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[McDermott, 80]. J. McDermott, "R1: An Expert in the Computer Systems Domain", Proc. American Association of Artificial Intelligence (AAAI), 1980, pp. 269-271.

[McDermott & Steele, 81]. J. McDermott and B. Steele, "Extending a Knowledge Based System to Deal With Ad hoc Constraints", Proc. Seventh Int'l. Joint Conf. on Artificial Intelligence (IJCAI-7), 1981.

[McDermott, 82]. J. McDermott, "XSEL: A Computer Salesperson's Assistant", in Machine Intelligence (vol. 10), J. Hayes, D. Michie, and Y. Pao (Eds.), Ellis Horwood, Chichester, England, 1982.

[Michie, 82]. D. Michie, "Expert Systems", Computer Journal, vol. 23, no. 4, 1982, pp. 369-376.

[Michie, 84]. D. Michie (Ed.), Introductory Readings in Expert Systems, Gordon and Breach, New York, NY, 1984.

[Miller, 73]. P. Miller, "A Locally-Organized Parser for Spoken Input", Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[Minsky, 75]. M. Minsky, "A Framework for Representing Knowledge", in The Psychology of Computer Vision, P. Winston (Ed.), McGraw-Hill, New York, NY, 1975, pp. 211-277.

[Moses, 71]. J. Moses, " Symbolic Integration: The Stormy Decade", Comm. of ACM, vol. 14, no. 8, 1971.

[Moses, 75]. J. Moses, "A MACSYMA Primer", Mathlab Memo No. 2, Computer Science Laboratory, MIT, Cambridge, MA, 1975.

[Mylopoulos, et al, 75]. J. Mylopoulos, A. Borgida, P. Cohen, and N. Roussopoulos, "TORUS - A Natural Language Understanding System for Data Management", Proc. Fourth Int'l Joint Conf. on Artificial Intelligence (IJCAI-4), 1975, pp. 414-421.

[Nau, 83]. D. Nau, "Expert Computer Systems", Computer , vol. 16, no. 2, 1983.

[Newell & Simon, 72]. A. Newell and H. Simon, Human Problem Solving, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Nii & Feigenbaum, 78]. H. Nii and E. Feigenbaum, "Rule-Based Understanding of Signals", in Pattern-Directed Inference Systems, D. Waterman and F. Hayes-Roth (Eds.), Academic Press, New York, NY, 1978.

[Nii & Aiello, 79]. H. Nii and N. Aiello, "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 645-655.

[Nii, et al, 82]. H. Nii, E. Feigenbaum, J. Anton, and A. Rockmore, "Signal to Symbol Transformation: HASP/SIAP Case Study", AI Magazine, vol. 3, no. 2, 1982, pp. 23-35.

[Nilsson, 71], N. Nilsson, Problem Solving Methods in Artificial Intelligence, McGraw-Hill, San Francisco, CA, 1971.

[Nilsson, 80]. N. Nilsson, Principles of Artificial Intelligence, Tioga Publishing Co., Palo Alto, CA, 1980.

[Osborn, 79]. J. Osborn, "Managing the Data from Respiratory Measurements", Medical Instrumentation, vol. 13, no. 6, 1979.

[Pople, 77]. H. Pople, "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning", Proc. Fifth Int'l. Joint Conf. Artificial Intelligence (IJCAI-5), 1977.

[Pople, 81]. H. Pople, "Heuristic Methods for Imposing Structure on Ill-structured Problems: The Structuring of Medical Diagnostics", in Artificial Intelligence in Medicine, P. Szolovitz (Ed.), Westview Press, Boulder, Colo., 1981, pp. 119-185.

[Popplestone, 67]. R. Popplestone, "The Design Philosophy of POP-2", in Machine Intelligence (vol. 3), D. Michie (Ed.), Edinburgh University Press, Edinburgh, 1967, pp. 393-402.

[Ralston, 76]. A. Ralston (Ed.), Encyclopedia of Computer Science, Van Nostard, New York, NY, 1976.

[Reddy, et al, 73]. R. Reddy, R. Fennell and R. Neely, "The HEARSAY Speech Understanding System: An Example of the Recognition Process", Proc. Third Int'l. Joint Conf. on Artificial Intelligence (IJCAI-3), 1973, pp. 185-193.

[Reddy, 75]. R. Reddy, (Ed.) Speech Recognition: Invited Papers of IEEE Symposium, Academic Press, New York, NY, 1975.

[Reiter, 81]. J. Reiter, "AL/X: An Inference System for Probabilistic Reasoning", Ph.D. Dissertation, Computer Science Department, University of Illinois, Urbana, IL, 1981.

[Rumelhart, 76]. D. Rumelhart, "Toward an Interactive Model of Reading", Technical Report No. 56, Center for Human Information Processing, University of California, San Diego, CA, 1976.

[Sacerdoti, 75]. E. Sacerdoti, "A Structure for Plans and Behavior", Technical Note 109, AI Center, SRI International Inc., Menlo Park, CA, 1975.

[Scott, et al, 77]. A. Scott, W. Clancey, R. Davis, and E. Shortliffe, "Explanation Capabilities of Knowledge-Based Production Systems", American Journal of Computational Linguistics, vol. 62, 1977.

[Shortliffe, 76]. E. Shortliffe, Computer-Based Medical Consultations: MYCIN, American Elsevier, New York, NY, 1976.

[Shortliffe & Buchanan, 75]. E. Shortliffe and B. Buchanan, "A Model of Inexact Reasoning in Medicine", Mathematical Biosciences, vol. 23, 1975, pp 351-379.

[Shortliffe, et al, 81]. E. Shortliffe, A. Scott, M. Bischoff, A. Campbell, W. Van Melle, and C. Jacobs, "ONCOCIN: An Expert System for Oncology Protocol Management", Proc. Seventh Int'l. Joint Conf. on Artificial Intelligence (IJCAI-7), 1981, pp. 876-881.

[Stallman & Sussman, 77]. R. Stallman, G. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", Artificial Intelligence, vol. 9, 1977, pp. 135-196.

[Stefik & Martin, 77]. M. Stefik and N. Martin, "A Review of Knowledge-Based Problem Solving as a Basis for a Genetics Experiment Design System", Computer Science Department, Heuristic Programming Project Memo HPP-75-5, Stanford University, Stanford, CA, 1977.

[Stefik, 78]. M. Stefik, "Inferring DNA Structures from Segmentation Data", Artificial Intelligence, vol. 11, 1978, pp. 85-114.

[Stefik, 80]. M. Stefik, "Planning with Constraints", Technical Report No. 784, Computer Science Department, Stanford University, Stanford, CA, 1980.

[Stefik, et al, 82]. M. Stefik, J. Aikins, R. Blazer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti, "The Organization of Expert Systems: A Tutorial", Artificial Intelligence, vol. 18, 1982, pp. 135-173.

[Sussman & McDermott, 72]. G. Sussman and D. McDermott, "From PLANNER to CONNIVER: A Genetic Approach", American Federation of Information Processing Society (AFIPS), 1972, pp. 1171-1180.

[Sussman, 77]. G. Sussman, "Electrical Design: A Problem for Artificial Intelligence Research", Proc. Fifth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-5), 1977, pp. 894-900.

[Swartout, 77]. W. Swartout, "A Digitalis Therapy Advisor With Explanations", Proc. Fifth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-5), 1977, pp. 819-825.

[Teitelman, 72]. W. Teitleman, "Do What I Mean (WHIM): The Programmer's Assistant", in Computers and Automation, W. Teitleman (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Teitelman, 78]. W. Teitelman, "INTERLISP Reference Manual", XEROX PARC, Palo Alto, CA, 1978.

[Tversky, 72]. A. Tversky, "Elimination by Aspects: a Theory of Choice", Psychological Review, vol. 79, 1972, pp. 281-299.

[Van Melle, 79]. W. Van Melle, "A Domain-Independent Producton Rule System for Consultation Programs", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 923-925.

[Van Melle, et al, 81]. W. Van Melle, E. Shortliffe, and B. Buchanan, "EMYCIN: A Domain-Independent System That Aids In Constructing Knowledge Based Consultation Programs", in Machine Intelligence (vol. 9), Infotech State of the Art Report 9, no. 3, Pergamon Infotech, Maidenhead, England, 1981.

[Weiss, 78]. S. Weiss, "A Model-Based Method for Computer-Aided Medical Decision-Making", Artificial Intelligence, vol. 11, no. 2, 1978.

[Weiss & Kulikowski, 79]. S. Weiss and C. Kulikowski, "EXPERT: A System for Developing Consultation Models", Proc. Sixth Int'l. Joint Conf. on Artificial Intelligence (IJCAI-6), 1979, pp. 942-947.

[Weiss, 81]. S. Weiss, "Expert Consultation Systems: The EXPERT and CASNET Projects", in Machine Intelligence (vol. 9), Infotech State of the Art Report 9, no. 3, Pergamon Infotech, Maidenhead, England, 1981.

[Weizenbaum, 66]. J. Weizenbaum, "ELIZA: A Computer Program for the Study of Natural Language Communication Between Man and Machine", CACM, vol. 9, no. 1, 1966, pp. 36-45.

[Wiederhold, 84]. G. Widerhold, "Knowledge and Database Management Systems", Software, vol. 1, no. 1, 1984.

[Winograd, 72]. T. Winograd, Understanding Natural Language, Academic Press, New York, NY, 1972.

[Winograd, 75]. T. Winograd, "Frame Representations and the Declarative/Procedural Controversy", in Representation and Understanding: Studies in Cognitive Science, D. Bobrow, A. Collins (Eds.), Academic Press, New York, NY, 1975, pp. 188-210.

[Winston, 77]. P. Winston, Artificial Intelligence, Addison-Wesley, Reading, MA, 1977.

[Wipke, et al, 77]. W. Wipke, H. Braun, G. Smith, F. Choplin, and W. Sieber, "SECS (Simulation and Evaluation of Chemical Synthesis): Strategy and Planning", in Computer Assisted Organic Synthesis, W. Wipke and W. House (Eds.), American Chemical Society, Washington, DC, 1977, pp. 97-127.

[Wolf & Woods, 80]. J. Wolf and W. Woods, "The HWIM Speech Understanding System", in Trends in Speech Recognition, W. Lea (Ed.), Prentice-Hall, Englewood Cliffs, NJ, 1980, pp. 310-339.

[Woods, 73a]. W. Woods, "An Experimental Parsing System for Transition Network Grammars", in Natural Language Processing, R. Rustin (Ed.), Algorithmics Press, New York, NY, 1973, pp. 111-154.

[Woods, 73b]. W. Woods, "Progress in Natural Language Understanding: An Application to Lunar Geology", American Federation of Information Processing Society (AFIPS), vol. 42, 1973, pp. 441-450.

[Yu, et al, 79]. V. Yu, L. Fagan, S. Wraith, W. Clancey, A. Scott, J. Hannigan, R. Blum, B. Buchanan, S. Cohen, R. Davis, J. Aikins, W. Van Melle, E. Shortliffe, and S. Axline, "Antimicrobial Selection for Meningitis by a Computerized Consultant - A Blinded Evaluation by Infectious Disease Experts", Journal of American Medical Association, vol. 241, 1979.

[Zadeh, 75]. L. Zadeh, "The Concept of a Linguistic Variable and its Application to Approximate Reasoning - I", Information Sciences, vol. 8, no. 3, 1975, pp. 199-249.

ABSTRACT

After being in a relatively dormant state for many years, only recently is artificial intelligence (AI) - that branch of computer science that attempts to have machines emulate intelligent behavior - accomplishing practical results. Most of these results can be attributed to the design and use of Knowledge-Based Systems, KBSs (or expert systems) - problem solving computer programs that can reach a level of performance comparable to that of a human expert in some specialized problem domain [Nau, 83]. These systems can act as a consultant for various requirements like medical diagnosis, military threat analysis, project risk assessment, etc. These systems possess knowledge to enable them to make intelligent decisions. They are, however, not meant to replace the human specialists in any particular domain.

In this thesis, a critical survey of recent work in interactive KBSs is reported, explaining KBS concepts and issues and techniques used to construct KBS. Application considerations to construct KBSs and potential future research areas in KBSs are identified.

A case study (MYCIN) of a KBS, a list of existing KBSs, and an introduction to the Japanese Fifth Generation Computer Project are provided as appendicies. Finally, an extensive set of KBS-related references are provided at the end of this report.

BIOGRAPHICAL SKETCH

Srinu Kavi was born in India on October 28, 1957. He attended Andhra University and Indian Institute of Science. He has a Masters degree in Physics from IISc, Bangalore, in 1980. He came to the University of Southwestern Louisiana in 1982, pursuing graduate studies in Computer Science, with a focus on Management Information Systems.

410

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
IN-82	~~44737~~ 183555	

4. Title and Subtitle

USL/NGT-19-010-900: KNOWLEDGE BASED SYSTEMS: A CRITICAL SURVEY OF MAJOR CONCEPTS, ISSUES, AND TECHNIQUES

5. Report Date DATE
December 11, 1984 OVERRIDE

6. Performing Organization Code

7. Author(s)

SRINU KAVI

8. Performing Organization Report No.

10. Work Unit No.

9. Performing Organization Name and Address

University of Southwestern Louisiana
The Center for Advanced Computer Studies
P.O. Box 44330
Lafayette, LA 70504-4330

11. Contract or Grant No.

NGT-19-010-900

13. Type of Report and Period Covered

FINAL; 07/01/85 - 12/31/87

12. Sponsoring Agency Name and Address

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This Working Paper Series entry presents a detailed critical survey of knowledge based systems. After being in a relatively dormant state for many years, only recently is artificial intelligence (AI) - that branch of computer science that attempts to have machines emulate intelligent behavior - accomplishing practical results. Most of these results can be attributed to the design and use of Knowledge-Based Systems, KBSs (or expert systems) - problem solving computer programs that can reach a level of performance comparable to that of a human expert in some specialized problem domain. These systems can act as a consultant for various requirements like medical diagnosis, military threat analysis, project risk assessment, etc. These systems possess knowledge to enable them to make intelligent decisions. They are, however, not meant to replace the human specialists in any particular domain. In this thesis, a critical survey of recent work in interactive KBSs is reported, KBSs are identified. A case study (MYCIN) of a KBS, a list of existing KBSs, and an introduction to the Japanese Fifth Generation Computer Project are provided as appendices. Finally, an extensive set of KBS-related references are provided at the end of this report.

This report represents one of the 72 attachment reports to the University of Southwestern Louisiana's Final Report on NASA Grant NGT-19-010-900. Accordingly, appropriate care should be taken in using this report out of the context of the full Final Report.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Knowledge-Based Systems, Information Storage and Retrieval Systems	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	306	

*For sale by the National Technical Information Service, Springfield, Virginia 22161